Innovaker Digital Circuits Kit

innovakêr

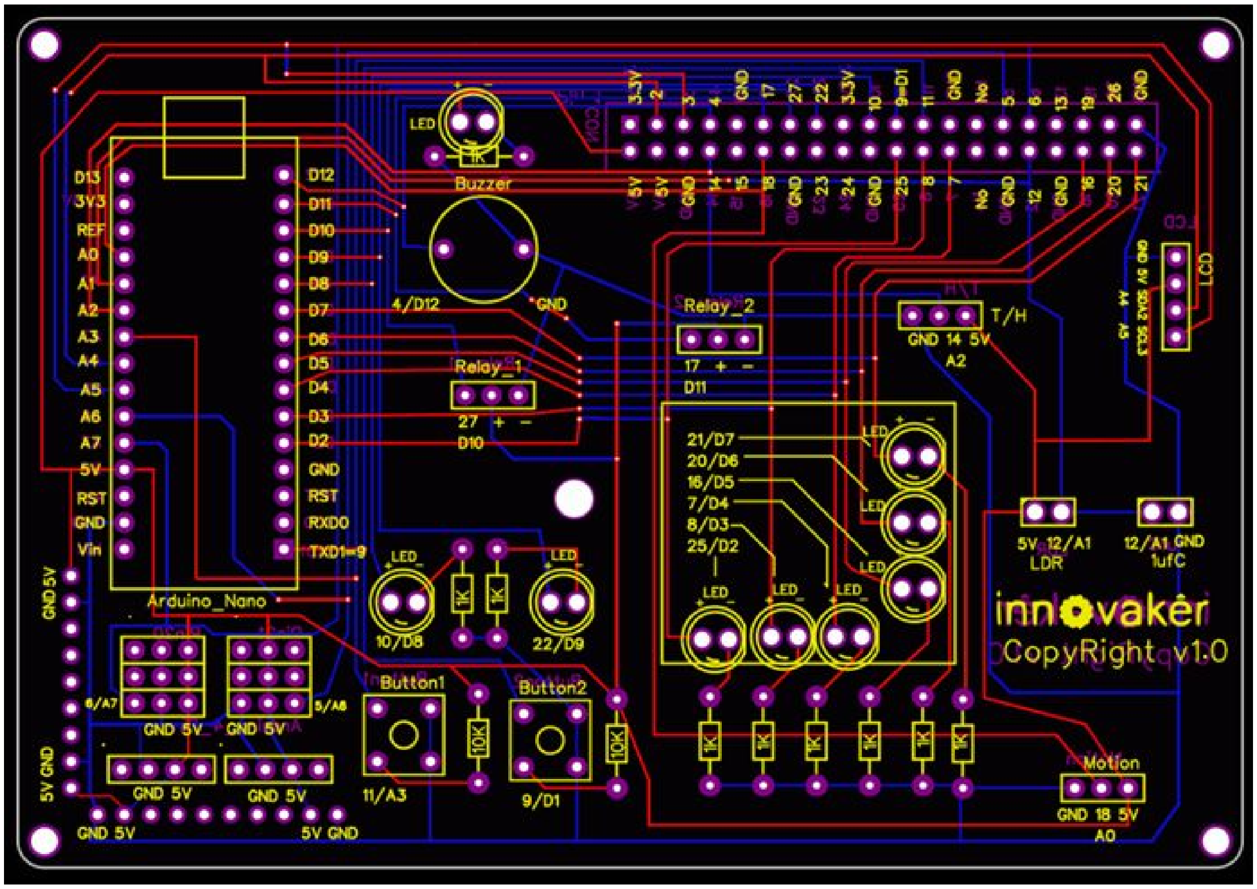
Content

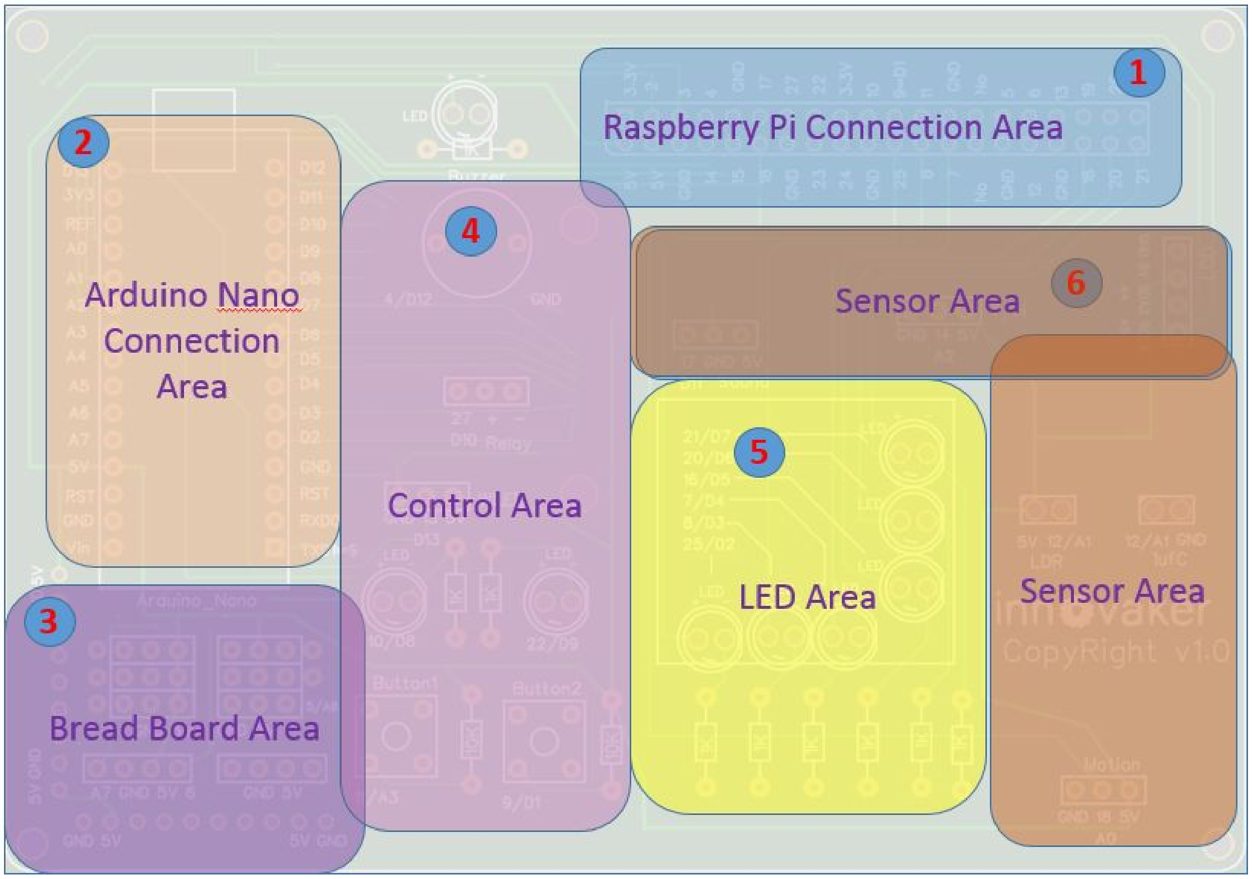
Introduction and Setup

Purpose

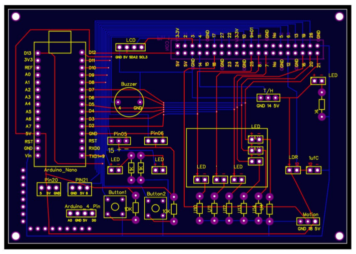
This book introduces the Raspberry Pi and electronic circuitry using the **Innovaker Digital Circuits Kit**. Using these tools, students can focus on electronic theory and programming instead of spending time making tedious wire connections. By undertaking a series of hands-on projects, students will learn how to develop rapid prototypes safely.

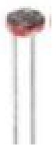
Innovaker Kit: What's inside?

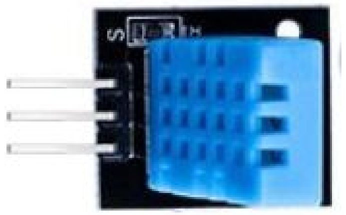
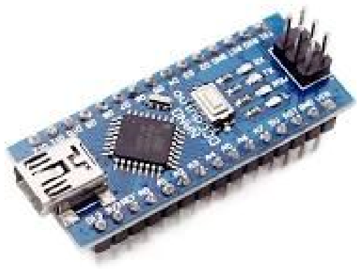




List

Name	Number	Price	Description	Picture
Pi PCB	1			

LED	16		4 each color color (RGBW)	
Resistor s	12		2 pcs 10kΩ 10 pcs 1kΩ	
LCD	1		4 pins connection	
Relay	1			
Buzzer	1			
Motion sensor	1			
Light sensor				
Button with cap	2			

1uf Capacitor	1			
DHT11 Temp & Humi Sensor	1			
Arduino Nano	1			
Package box				
Pins				
Labor				
Total				

Electricity: Voltage, Current, Resistance

In order to understand circuits, we must first define the quantities that define how electricity flows.

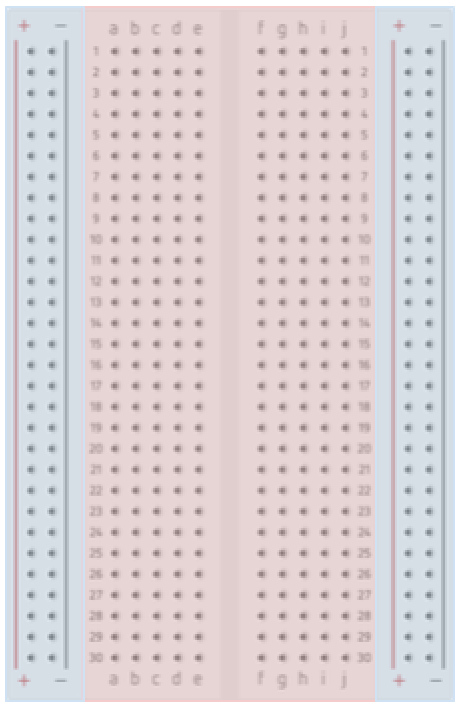
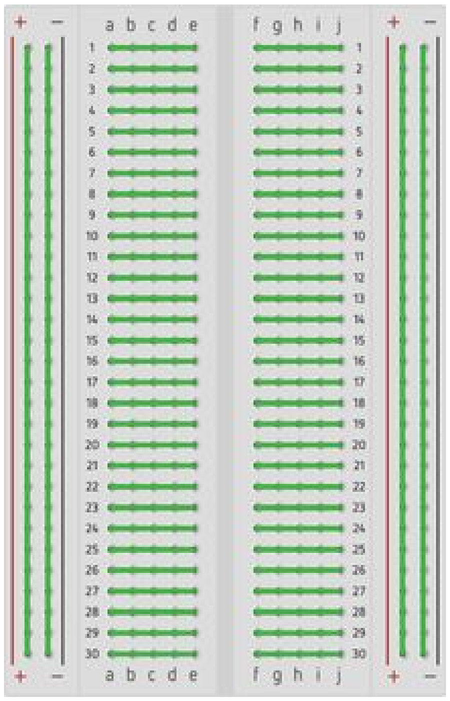
Can you add some picture to explain these?

Charge (Q) is the basic unit of measurement that electricity is built upon.

Voltage (V) is the difference in potential energy between two points along a circuit and is measured in the units of volts(V). Because calculating the voltage requires two points, one node is typically chosen as a reference to which all other voltages can be calculated with respect to. This reference is called the **ground**.
Current (I) is the amount of charge that flows through across a period of time and is measured in the units of amperes(A).
Resistance (R) describes the amount of obstruction that we have to the flow in the circuit and is measured in the units of ohms(Ω). Given two circuits with an equal amount of voltage, the one with more resistance will have less current flowing through it.
To better imagine these abstract concepts, think of pipe of flowing water. In this case, the charge in our circuit is similar to the water. Like the pressure that causes water to flow in this tank, voltage is what drives the charge in our current to move. The amount of water that passes through this pipe over a certain period of time is similar to the concept of current, except that it is a measure of the charge that flows through. Lastly, imagine that there is now a small dent in our pipe. The narrow path impedes the flow of water, much like resistance impedes the flow of electricity.

Breadboards

Breadboards are one of the fundamental tools that we must learn how to use before building circuits. Because they allow us to make electrical connections without having to permanently solder components together, they allow for the rapid prototyping of circuits.

Each row of the breadboard is numbered from 1-30, while each column are lettered a-j. Internally, each of the rows are connected with conductive metal strips that allows electricity to flow. The middle chasm, between the ‘e’ and the ‘f’ divides each row into two parts. This middle section, the assembly area, is where we can form connections between different components.

On both the right and left sides, you will find two long power rails, the red column connected to the + and the blue column connected to the —. Instead of being connected as rows, however, these rail are connected beneath the surface as columns. Each of these, known as the power rails, are intended to supply a power source and ground to our components

Electrical Diagrams

While breadboards show us how things are being placed physically, they aren’t always a good depiction of the electrical connections that drive circuits. For this, we turn to electrical diagrams/schematics, which establish how each component is being wired in relation to one another. Throughout this course, we will be learning both how to analyze these diagrams and physically create the circuits that they represent.

(PICTURE HERE)

Multimeter

Multimeters are a tool that allow you take take multiple measurements, including that of voltage, current, resistance, and more. In order to use a multimeter, connect the device with two probes, typically one of which is red and the other black. Connect the black probe to the COM port and the red to the mAVΩ port. Turn the dial to whichever function you are looking for and follow the following guide to correctly find the value which you are looking for.

Raspberry Pi 3:

The Raspberry Pi 3 is a mini computer the size of a credit card. Much like many people use Windows or macOS as an operating system on their personal computers, the Raspberry Pi 3 comes preinstalled with its own operating system, a version of Linux called Raspbian. Many assume that computers require a monitor, keyboard, and mouse in order to fully utilize. This may not necessarily be true and we will take a look at the various different ways that we can interface with our Raspberry Pi.

Methods of Connecting:

Monitor, Keyboard, and Mouse

The most standard way of connecting with our Raspberry Pi is using a monitor, keyboard, and mouse. Plug in a monitor through the Raspberry Pi's HDMI port and the keyboard/mouse through the USB A ports. Upon supplying the Raspberry Pi with power, you will immediately see a user friendly interface. At this point, your Raspberry Pi will be able to do almost everything your personal computer can: access the Internet, create documents, play games, and more!
Use this way to set up wifi connection, reboot the PI from the pulldown menu "Shutdown", before you restart without monitor, you can put LCD in the kits, after Raspberry Pi starts, the IP address will be displayed on LCD as following picture:

Internet connected	Internet not connected

SSH

Secure Shell (SSH) is a secured network protocol that allows for a secure remote login from one computer to another. We can use SSH to create tunnels between your personal computer and the Raspberry Pi, allowing us to operate the Raspberry Pi as a headless device. In order to utilize the device, users must be familiar with the terminal command line to execute commands.

Getting Set Up:
Windows:

Set Up

MacOS or Linux:
Use the `ssh` command to SSH into a device using the IP Address
```
ssh <username>@<ip-address>
```
Ex. ssh pi@193.24.1.15

VNC

Virtual Network Computing (VNC) is another method of remotely connecting with a device, but uses a much more familiar graphical user interface. When connected with the Raspberry Pi, all mouse and keyboard events from the remotely connected device are transmitted to the Pi, and the graphical interface is updated accordingly.

Getting Set Up:
Install RealVNC Viewer on your local computer.
https://www.realvnc.com/en/connect/download/viewer/

Connect to the Raspberry Pi its IP address

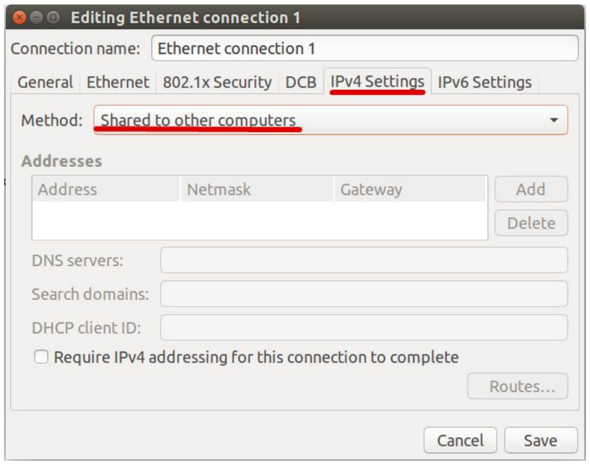
Setting Up a headless Raspberry Pi using an Ethernet cable:

Here avi to show how to connect through Window:
https://www.youtube.com/watch?v=AJ7skYS5bjI

The following guide describes how to set up your headless Raspberry Pi using the included SD card in the Innovaker kit. The process will require a stable Wifi connection and an Ethernet cable.

1. Use the ethernet cable to connect your computer's ethernet port to that of the Raspberry Pi's
2. Go to your computer's network connection settings
3. Linux
    a. Create a new Ethernet connection
    b. Go to IPv4 settings
    c. Select "Shared to other computers" as the method of connection
4. Windows
    a. Click on your current Wifi connection
    b. Access the Properties
    c. Select the sharing tab and enable both settings

5. Save those settings


Editing Ethernet connection 1

Connection name: Ethernet connection 1

General | Ethernet | 802.1x Security | DCB | IPv4 Settings | IPv6 Settings

Method: Shared to other computers

Addresses

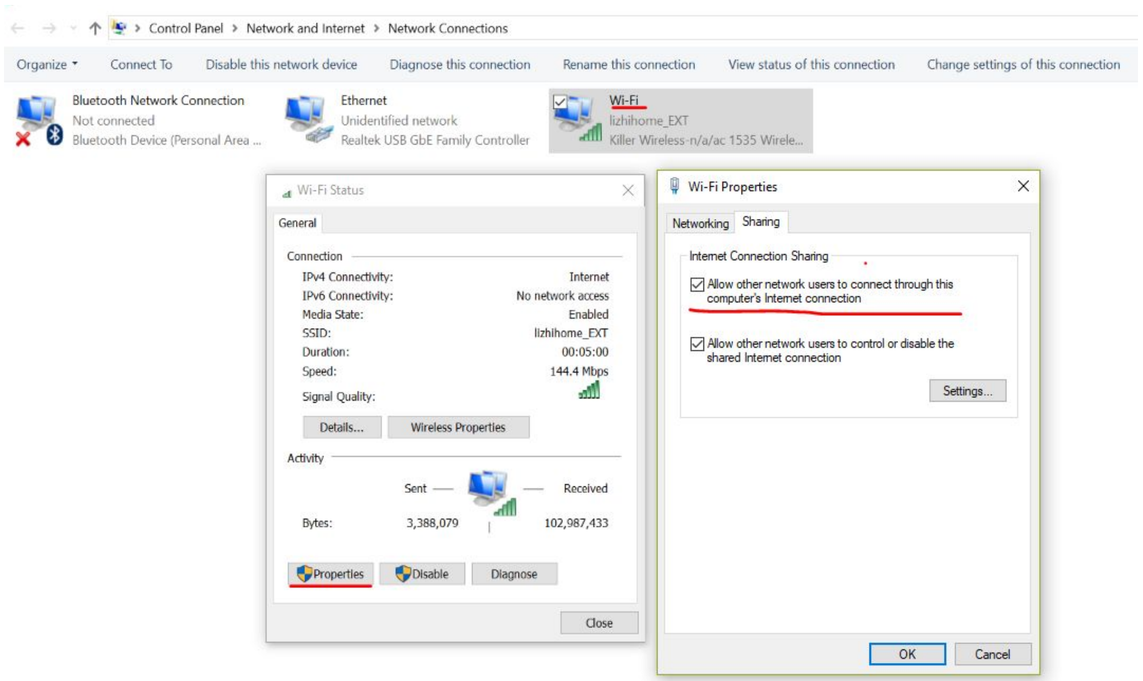
Address	Netmask	Gateway

DNS servers:

Search domains:

DHCP client ID:

☐ Require IPv4 addressing for this connection to complete


Control Panel > Network and Internet > Network Connections

Organize
Connect To
Disable this network device
Diagnose this connection
Rename this connection
View status of this connection
Change settings of this connection

Bluetooth Network Connection
Not connected
Bluetooth Device (Personal Area ...

Ethernet
Unidentified network
Realtek USB GbE Family Controller

Wi-Fi
lizhihome_EXT
Killer Wireless-n/a/ac 1535 Wirele...

Wi-Fi Status

General

Connection

IPv4 Connectivity:	Internet
IPv6 Connectivity:	No network access
Media State:	Enabled
SSID:	lizhihome_EXT
Duration:	00:05:00
Speed:	144.4 Mbps
Signal Quality:	

Activity

	Sent	Received
Bytes:	3,388,079	102,987,433

Wi-Fi Properties

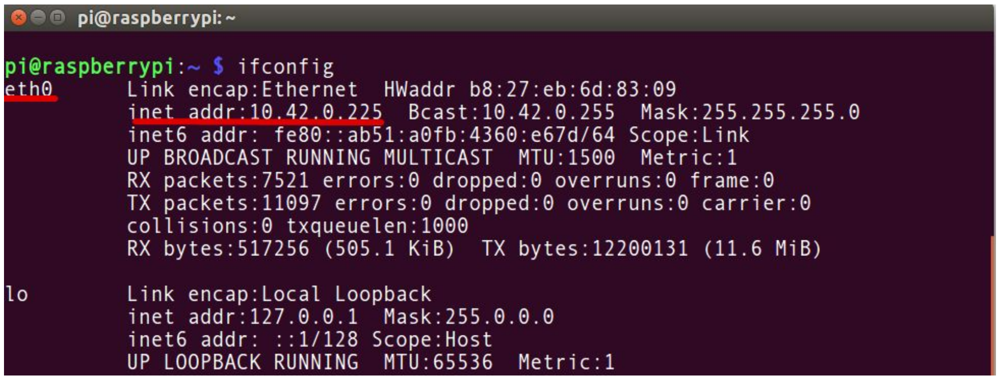
Networking | Sharing

Internet Connection Sharing

☑ Allow other network users to connect through this computer's Internet connection

☑ Allow other network users to control or disable the shared Internet connection

6. User either the terminal or PuTTY to SSH into the `pi@raspberrypi.local` address
7. Use the `ifconfig` or `ipconfig` to determine the local ethernet IP address. We will be using this temporary IP address to setup a stable Wifi connection directly from the Raspberry Pi



```
pi@raspberrypi:~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:6d:83:09
          inet addr:10.42.0.225  Bcast:10.42.0.255  Mask:255.255.255.0
          inet6 addr: fe80::ab51:a0fb:4360:e67d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7521 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11097 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:517256 (505.1 KiB)  TX bytes:12200131 (11.6 MiB)

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
```

8. Download the VNC Viewer application and connect to the device with this ethernet IP address. At this point, you should be able to see a visual interface.
9. Click on the Wifi symbol at the top and connect to your local Wifi network
10. Unplug your Raspberry Pi from your computer.
11. Using either `ifconfig` or `ipconfig` once again, you should be able to see that the device is now connected to the wireless network and is given a new ip address of its own. You can use VNC or SSH to access your device remotely from this point forward.

Python:

1 Install and Running Python Program

Python is a general purpose and powerful programming language. Here we will use Python program to work with a Raspberry Pi.

We use Python 3.

Python has been pre-installed in the micro flash card which is attached with the Innovaker Digital Circuits Kit. The Python example code for a Raspberry Pi to control digital circuits has been attached as well.

There are two ways to running Python program:

1) Typing commands directly in a Python shell window
2) Run a program stored in a file with file type as .py

2 Python Basics

2.1 Basic Data Types

int – integer, e.g. 1, 2, 100, 123
float – decimal number, e.g. 3.14, 10.5, 0.125
string – e.g. “Hello!”, “Tom Smith”, “”, ‘school’, ‘USA’

2.2 Variables

Variable(Identifier) - represent data storage in memory

2.2.1 Naming rules

- Can contain letters, digits and underscores (_)
- Must start with letter or underscore (_)
  e.g “my_number2”, “_myDog” are valid
- Can't start with a number
  e.g “5_keys” is not valid
- It’s case sensitive
- Can be of any length
- Can't be a Python keyword
- No hyphens in the middle
- No spaces are allowed

2.2.2 Assign Value to a Variable and Expression

In Python no need to declare types of variable. Interpreter auto detects the type of a variable by the data it contains.
Equal sign (=) is used to assign value to a variable.
= is also known as assignment operator

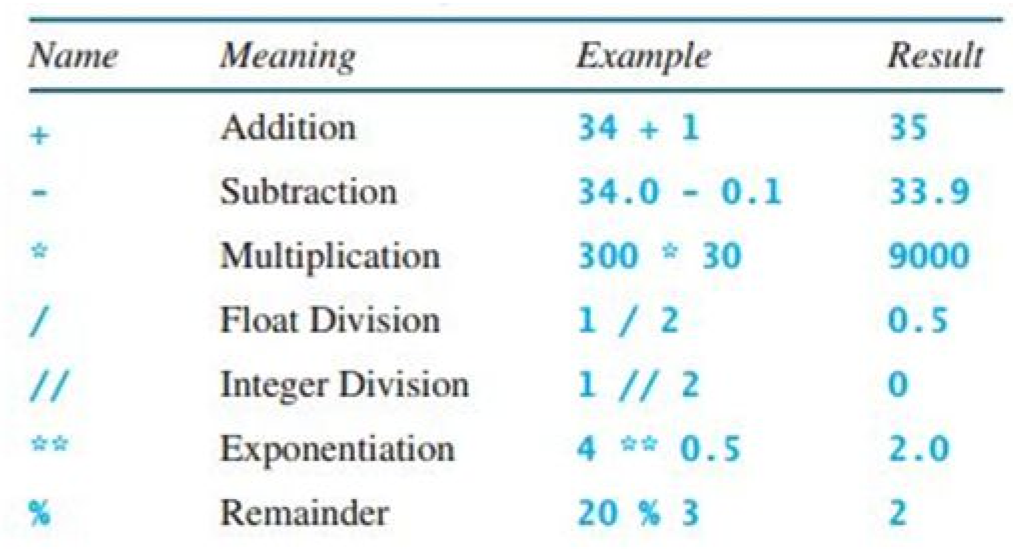
Expression is like

```
a = 10
b = 20
c = a + b + 100

number1 = 11
number2 = 12
number3 = 13

average_number = ( number1 + number2 + number3 ) / 3.
```

2.3 Python Numbers and Operators

Python 3 Support 3 Different Numerical Types:

Int	- for integer values like 1, 2, 30, 100
Float	- for floating point values like 3.14, 12.5, 0.125
Complex	- for complex numbers like 3 + k + m, x**2 + y**2



Name	*Meaning*	*Example*	*Result*
+	Addition	`34 + 1`	35
-	Subtraction	`34.0 - 0.1`	33.9
*	Multiplication	`300 * 30`	9000
/	Float Division	`1 / 2`	0.5
//	Integer Division	`1 // 2`	0
**	Exponentiation	`4 ** 0.5`	2.0
%	Remainder	`20 % 3`	2

+ , - and * works as expected

/ - Float Division : / operator returns result as floating point number
E.g.

```
>>> 3/2
1.5
```

// - Integer Division : // operator removes the decimal part of the result and returns only integer.

```
>>> 3//2
1

>>> 2//3
0
```

** - Exponentiation Operator : compute ab

```
>>> 2 ** 3      # is same as 2 * 2 * 2
8
```

% operator : also known as remainder operator.
This operator returns remainder of the result of division.

E.g.:

```
>>> 7 % 2
1
```

Augmented Assignment Operator

These operator allows you write shortcut assignment statements.

e.g:

```
>>> count = 1
>>> count = count + 1
>>> count
2
```

by using Augmented Assignment Operator we can write it as:

```
>>> count = 1
>>> count += 1
>>> count
2
```

2.4 String

Create strings

```
name = "tom"  # a string
mychar = 'a'    # a character
```

String Operations

String index starts from **0**
So to access the first character in the string type is like:

```
>>> name = “Mike”
>>> name[0]
M
>>> name[1]
i
>>> name[2]
K
>>> name[3]
e
```

3 Python Control Statement

Comparing Operators

Symbol	Description
<=	smaller than or equal to
<	smaller than
>	greater than
>=	greater than or equal to
==	equal
!=	not equal to

The result of comparison will always be a boolean value True or False.

if Statement

```
if boolean-expression:
        #statements 1
else:
        #statements 2
```

4 List, Tuple and Dictionary

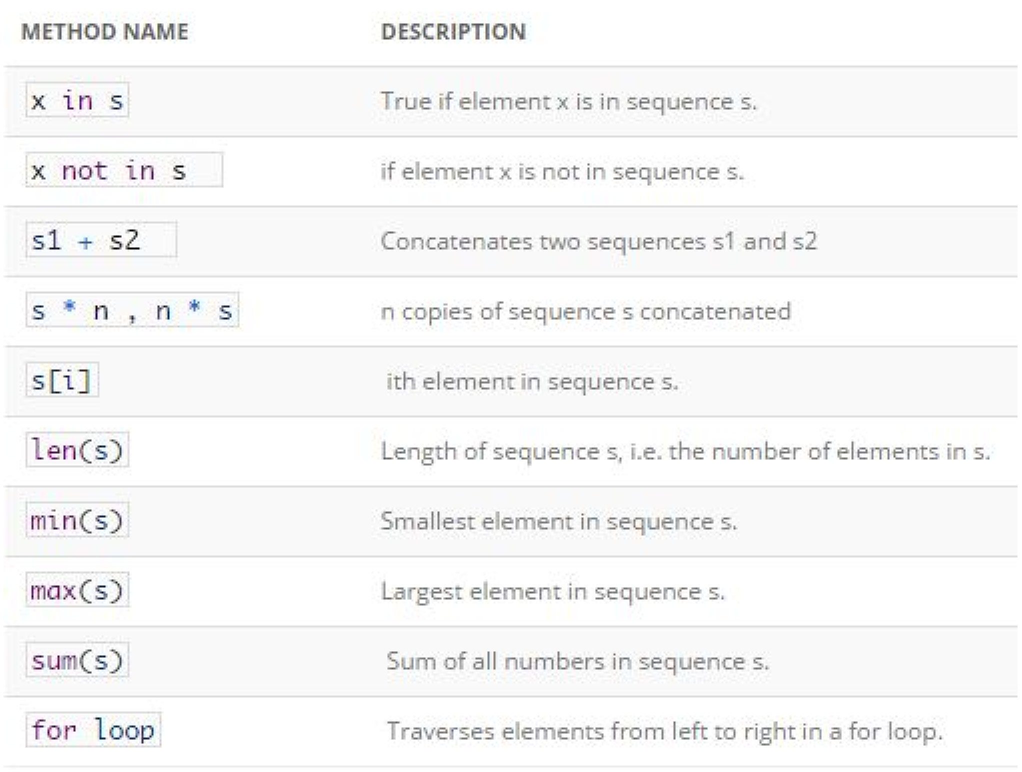
Python List

```
list1 = [1, 2, 3, 4]   #can't be list = [1,2,3,4] as "list" is Python keyword

list2 = [ "a", "b", "c", "d" ]
```

Accessing Elements in List

```
>>> list_1 = [1,2,3,4,5]
>>> list_1[0]
1
>>> list_1[1]
2
>>> list_[4]
5
```

List Common Operators



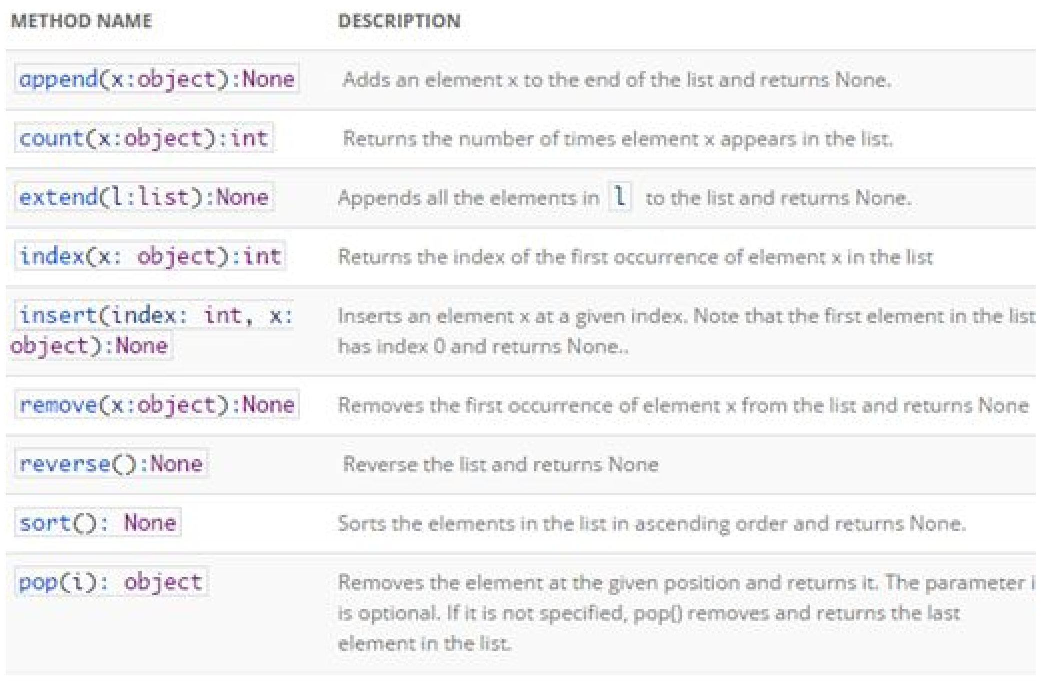
METHOD NAME	DESCRIPTION
`x in s`	True if element x is in sequence s.
`x not in s`	if element x is not in sequence s.
`s1 + s2`	Concatenates two sequences s1 and s2
`s * n , n * s`	n copies of sequence s concatenated
`s[i]`	ith element in sequence s.
`len(s)`	Length of sequence s, i.e. the number of elements in s.
`min(s)`	Smallest element in sequence s.
`max(s)`	Largest element in sequence s.
`sum(s)`	Sum of all numbers in sequence s.
`for loop`	Traverses elements from left to right in a for loop.

in or not in operator

List Function Examples

```
>>> list1 = [2, 3, 4, 1, 32]
>>> 2 in list1
True
>>> 33 not in list1
True
>>> len(list1) # find the number of elements in the list
5
>>> max(list1) # find the largest element in the list
32
>>> min(list1) # find the smallest element in the list
1
>>> sum(list1) # sum of elements in the list
42
```

Commonly Used list Methods with Return Type



METHOD NAME	DESCRIPTION
`append(x:object):None`	Adds an element x to the end of the list and returns None.
`count(x:object):int`	Returns the number of times element x appears in the list.
`extend(l:list):None`	Appends all the elements in `l` to the list and returns None.
`index(x: object):int`	Returns the index of the first occurrence of element x in the list
`insert(index: int, x: object):None`	Inserts an element x at a given index. Note that the first element in the list has index 0 and returns None..
`remove(x:object):None`	Removes the first occurrence of element x from the list and returns None
`reverse():None`	Reverse the list and returns None
`sort(): None`	Sorts the elements in the list in ascending order and returns None.
`pop(i): object`	Removes the element at the given position and returns it. The parameter i is optional. If it is not specified, pop() removes and returns the last element in the list.

List Method Examples

```
>>> list1 = [2, 3, 4, 1, 32, 4]
>>> list1.append(19)
>>> list1
[2, 3, 4, 1, 32, 4, 19]
>>> list1.count(4) # Return the count for number 4
2
>>> list2 = [99, 54]
>>> list1.extend(list2)
>>> list1
[2, 3, 4, 1, 32, 4, 19, 99, 54]
```

4 Python Loops

LED

What is a LED?

An LED, or light emitting diode, is an electrical component that converts electrical energy into light. LED’s come in many shapes and forms, but the ones we will be using emit only a single color when current is passed through them.

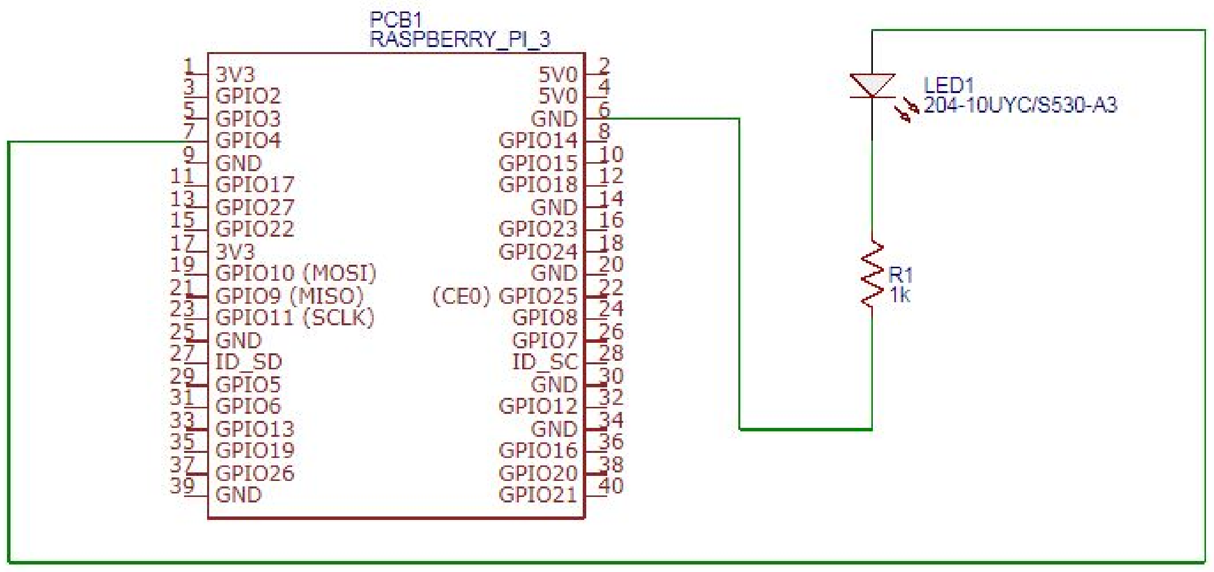
How does it work?

LEDs are diodes, meaning that they only allow current to flow through them in a single direction. Looking at an LED, we see that it has two legs, one of which should be longer than the other. The positive end will always be connected to the longer leg, known as the **anode**. The negative end is then the shorter leg, known as the **cathode**.

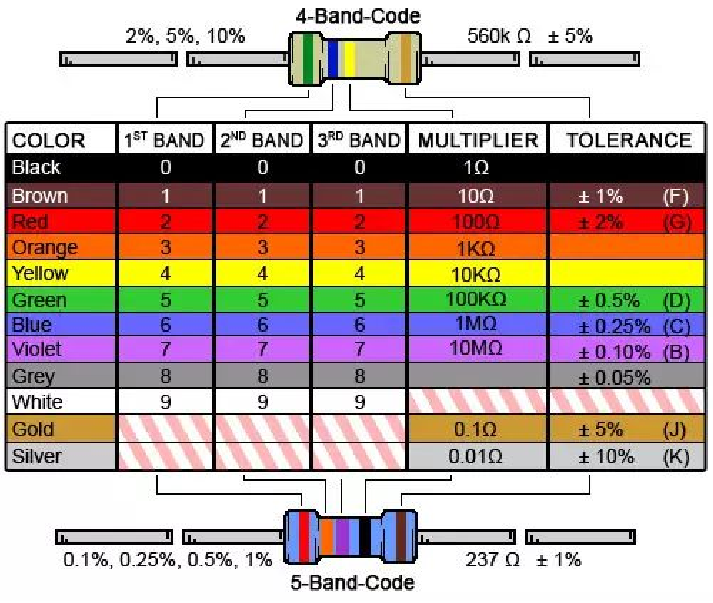
LEDs themselves have a limit on the amount of current that can pass through them before they burn out and become dysfunctional. For this reason, it is generally advised not to connect your LEDs directly to your source of power. Rather, use a resistor to limit the amount of current that is passing through the LED so that it doesn’t blow out.

WARNING: Make sure when connecting your LEDs that the positive end is connected to the positive end of the power source and the negative end is connected to the path that leads to ground. Failure to do so may result in a damaged LED.

Building the Circuit



Resistor Color Code


4-Band-Code
2%, 5%, 10%
560k Ω ± 5%
0.1%, 0.25%, 0.5%, 1%
237 Ω ± 1%
5-Band-Code

COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9	▨	▨
Gold	▨	▨	▨	0.1Ω	± 5% (J)
Silver	▨	▨	▨	0.01Ω	± 10% (K)

All resistors are standardized and can be identified by the bands of colors on them. Look at the following diagram in order to determine the resistance of your resistor. For example, the 4-Band-Code below, with the color sequence of “Green-Blue-Yellow” corresponds to the number 56 and a multiplier of 10k. Thus, we can determine the resistance of this resistor to be 56 * 10 x 10^3 = 560k Ω

Writing the Code

Run the following Python program in order to allow the LED to blink. Please take note of the structure of the program, which we will attempt to emulate across all our programs for its good programming practices.

`led.py`

```
import RPi.GPIO as GPIO
import time
```

The top of our file will contain all of the libraries we import. Here, we import the `GPIO` and `time` modules

```
LED_PIN = 10
ON = True
OFF = False
```

Constants typically do not change in value and are given names in all uppercase.
Pin numbers should always be defined as constants.

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_PIN, GPIO.OUT)
```

The `setup` method is where we define the purpose of our GPIO pins.

`GPIO.setmode(GPIO.BCM)` - Sets up our board so that the GPIO pins can be read according to BCM standards

`GPIO.setup(LED_PIN, GPIO.OUT)` - Sets this specific GPIO pin to be used as an output

```
def main():
    while True:
        GPIO.output(LED_PIN, ON)
        time.sleep(1)
        GPIO.output(LED_PIN, OFF)
        time.sleep(1)
```

The `main` method controls the logic of our program. Here, we use a while statement to turn our light on and off indefinitely.

`GPIO.output(LED_PIN, ON)` - turns the GPIO pin on

`time.sleep(1)` - halts our program for a period of 1 second

```
if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

Lastly, we will end with this block, which checks whether program is being run directly. Inside, we run the two methods that we defined earlier, and we use the `GPIO.cleanup()` to clear all output from the GPIO pins upon finishing our program.

To ensure that your program is readable to both computer and human eyes, make sure that you structure your programs in a way that coherent on what the purpose of the program is. This will also eliminate many of the bugs that occur in your code.

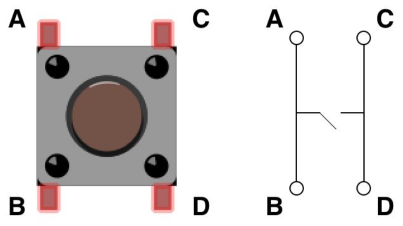
Buttons and Buzzers

What is a Button?

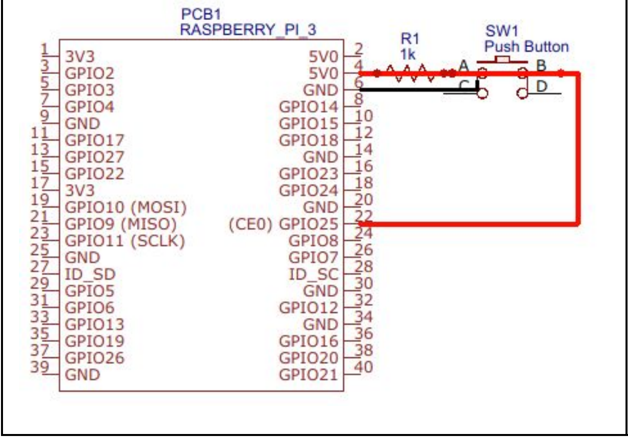
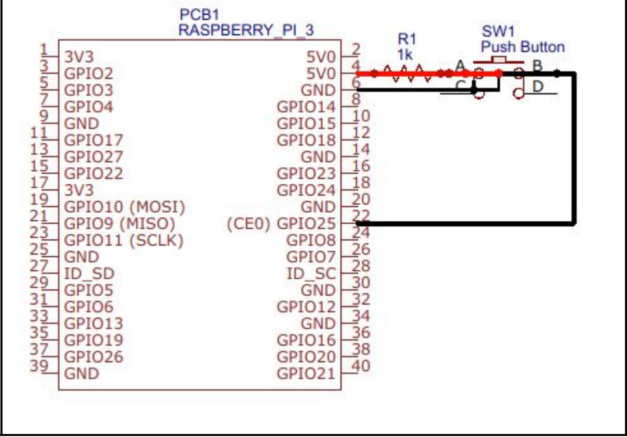
Buttons can be seen almost everywhere in the world around us in elevators, phones, computers, and much more. Buttons are simply a form of a switch which changes state when it is pressed. Buttons allow us to detect physical changes and convert them into electrical signals.

How does it work?

If you hold the button such that it is in a vertical orientation, then the two rails that are parallel to each other are connected internally. When the button is pressed down, the switch inside closes, connecting the two rails. Current from one branch can then flow across to the other.

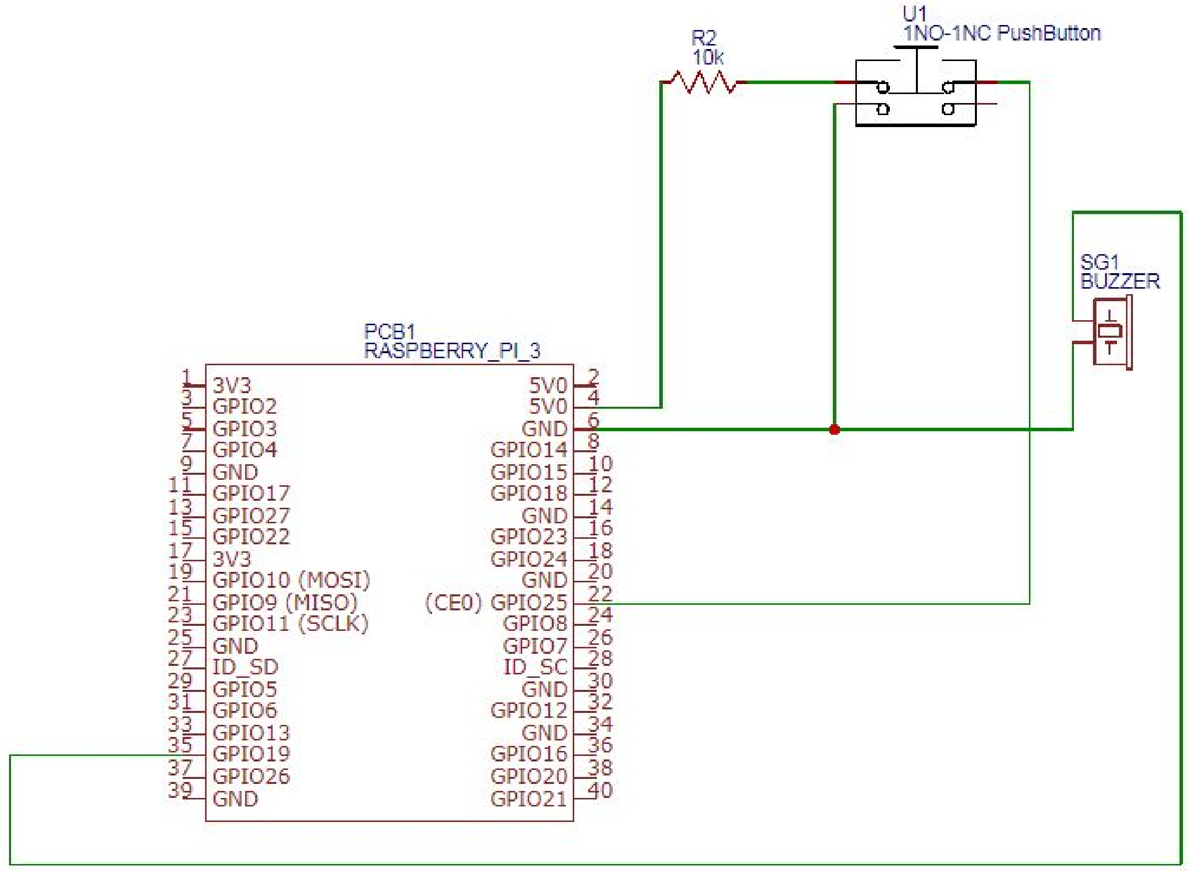


In order to detect a state change from the button, we will be connecting a 5V power source to the A terminal, a GPIO pin to the B terminal and ground to the C terminal. When the button is in its natural, unpushed state, our GPIO pin will detect the 5V signal as an input. Upon being pushed, however, all four terminals become connected, causing our GPIO pin to be linked with the ground. When it is read in this state, the input that being read will equal that of the ground. Because our button requires very little current in order to function, use a resistor with larger resistance magnitude, either 1k Ω or 10k Ω.

There are two different ways that we can read a GPIO input from a button. The first method, called **polling**, involves continuously checking for an input using a loop. This can potentially be faulty if the input is read at the wrong time and can be processor intensive at times. The other method is using **interrupts**, which listens for changes in the state of the buttons rather than what the state is.

Building the Circuit



Writing the Code

Run the following programs that sound a buzzer whenever a button is pressed. Take note of the two different ways in which we can listen for button presses.

`button1.py`

```
import RPi.GPIO as GPIO
import time

BUTTON_PIN = 11
BUZZER_PIN = 4
ON = True
OFF = False

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BUTTON_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_UP)
    GPIO.setup(BUZZER_PIN, GPIO.OUT)
```

Notice how the button setup differs slightly from that of the other pins.

`GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)` - the button setup takes a third parameter, a function, to determine which state the button is naturally in.

`GPIO.input(BUTTON_PIN)` - Reads the status of the GPIO pin. When the `pull_up_down` status is `GPIO.PUD_UP`, the natural state is a 1 while a push is a 0

```
def main():
    while True:
        time.sleep(0.01)
        if(GPIO.input(BUTTON_PIN) == 0):
            print ("Button is pressed!")
            print ("Buzzer will be turn on!")
            GPIO.output(BUZZER_PIN, ON)
        else:
            GPIO.output(BUZZER_PIN, OFF)

if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

`button2.py`

```
import RPi.GPIO as GPIO
import time

BUTTON_PIN = 11
BUZZER_PIN = 4
ON = True
OFF = False

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BUTTON_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)
    GPIO.add_event_detect(BUTTON_PIN,
GPIO.BOTH, callback=button_callback,
bouncetime=200)
    GPIO.setup(BUZZER_PIN, GPIO.OUT)

def main():
    while True:
        print("I'm working...")
        time.sleep(0.01)
        GPIO.setup(BUZZER_PIN, GPIO.LOW)
```

This version of attaches a **callback** function as a parameter to the `GPIO.add_event_detect` function. The callback runs whenever an event is detected from the button. The callback that we pass in here is defined below.

```python
def button_callback(channel):
    print("button pressed!")
    BuzzerStatus = GPIO.input(BUZZER_PIN)
    if BuzzerStatus:
        GPIO.output(BUZZER_PIN, GPIO.LOW)
    else:
        GPIO.output(BUZZER_PIN, GPIO.HIGH)

if __name__  == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

This is the callback function that is passed in to the event detection in the setup phase. This function is run whenever our button detects a change in events.

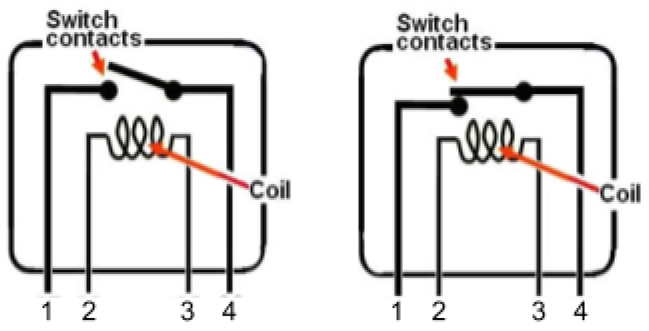
Relays

What is a relay?

Relays are a type of electromagnetic switch that uses a small amount of current to determine its state. Many of the times, we will find ourselves working with appliances that contain a much higher current than the electronic equipment. In order to control the state of this much higher current using our Raspberry Pi, we need a relay in order to bridge the gap between the two. For example, imagine that we want to use the GPIO pins of the Raspberry Pi to control current flowing from the wall socket. We can't directly connect the wall socket to a switch because our Raspberry Pi can't handle that much voltage passing through it and can potentially break. The relay adds an alternative "high current" route that does not affect the current flowing through the Raspberry Pi but is still dependent on the "low current" generated by our Raspberry Pi's pins.

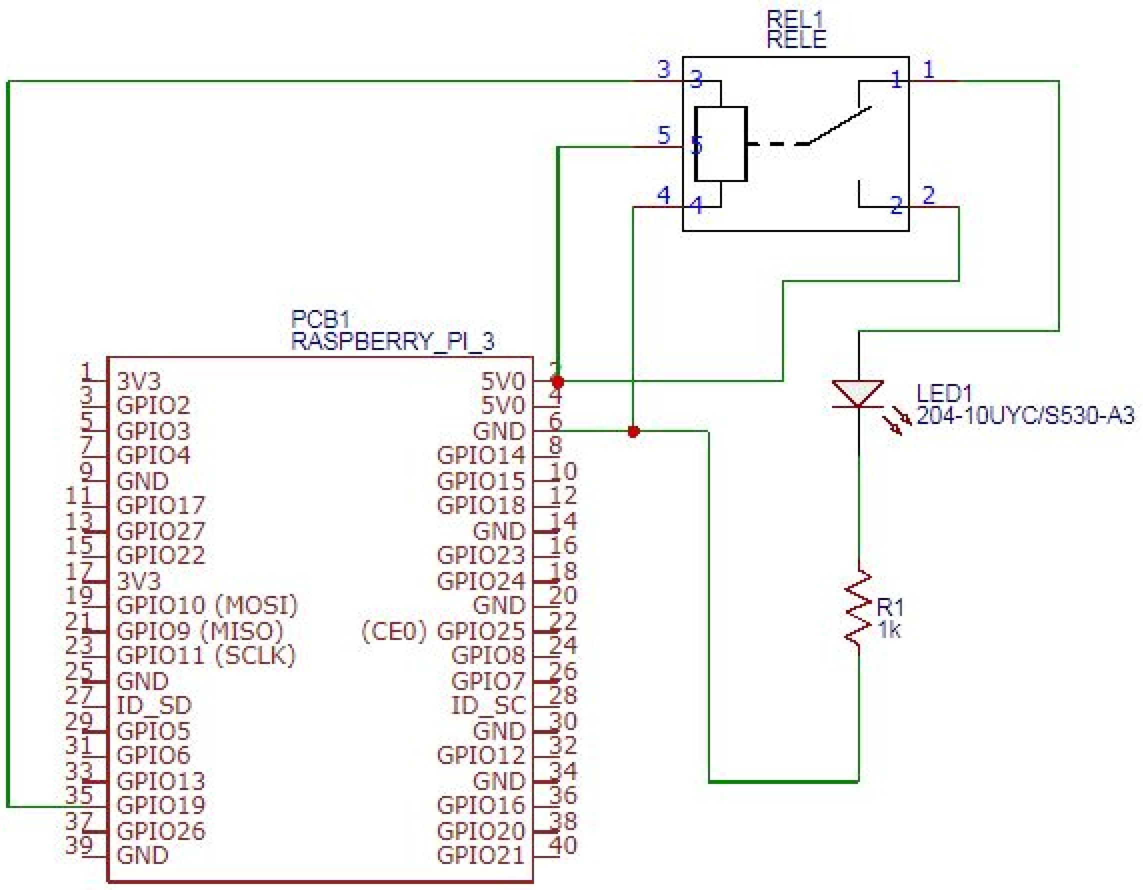
How does it work?

All relays will have 4 terminals, forming pairs for the high current and low current routes. Internally, the low current route (2 & 3) is connected to an electromagnetic coil and the high current route (1 & 4) is connected to a switch. When the low current is passed through the coil, it causes a force to be applied onto the switch contact of the high current path, closing the switch and allowing the high current to flow.



WARNING: When forming connections to the different terminals of the relay, make sure that you identify and separate the low and high current routes. Failure to do so may potentially cause your Raspberry Pi to break.

Building the Circuit



Writing the Code

The following program causes a relay to turn off and on. When correctly assembled, a distinct clicking noise should be heard from the relay.

relay.py

```
import RPi.GPIO as GPIO
import time

RELAY_PIN = 27
ON = True
OFF = False

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(RELAY_PIN, GPIO.OUT)

def main():
```

Notice how the relay is set up in much the same way we would program an LED as an output pin.

Using the `GPIO.output(PIN,STATE)` function

```
    while True:
        GPIO.output(RELAY_PIN, ON)
        time.sleep(2)
        GPIO.output(RELAY_PIN, OFF)
        time.sleep(2)

if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

will allow us to turn our relay either on or off

LCD Displays

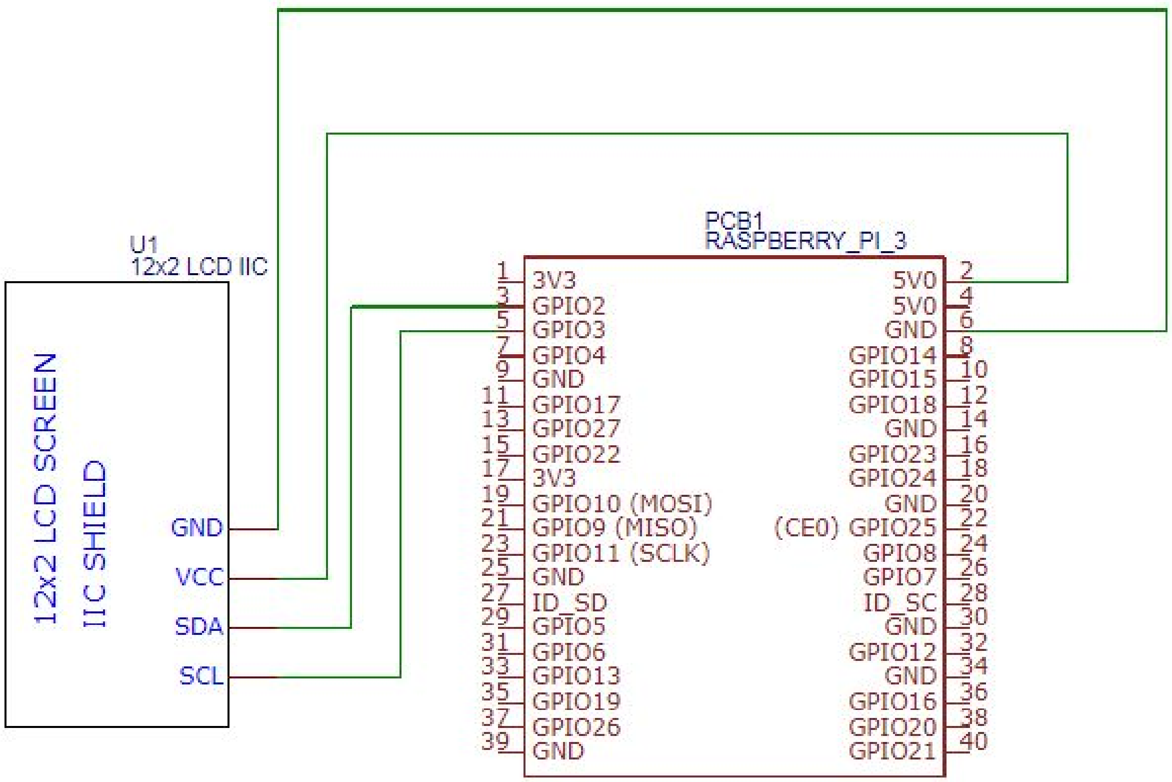
What is a LCD?

The **Liquid Crystal Display (LCD)** is a type of screen that uses liquid crystals in order to create a display. A backlight is shown through a layer of liquid crystals, which produces either a colorized or monochrome color. When large arrays of them are put together, they can form large panels to display images or text. As a result, LCDs can be found all around us today — in our computers, cell phones, televisions, microwaves, clocks, and much more.

How does it work?

The LCD we will be using is a 16x2 screen that is enabled with I2C communication, allowing us to effectively use only 4 pins to display words onto our screen. Two of the pins are connected to 5V and ground in order to supply power to the device, while the other two pins are used to send data and communicate with the device.

Building the Circuit



Writing the Code

The code we will be writing is based on a prewritten library to streamline the process of using our display. The following program displays the current time on the screen and refreshes every minute.

LCD.py

`import RPi.GPIO as GPIO`
`import time`
`from led4import import *`
`from datetime import datetime`
	Here we have two additional imports
	`led4import` - LCD library
	`datetime` - python library for dates and time

`def setup():`
`    lcd_init()`
	`lcd_init()` - function to initialize the LCD

`def main():`
`       while True:`
	`lcd_string(STRING, LINE #)` - function to display strings on the LCD. Takes two parameters - the string to display and the line

`lcd_string(str(datetime.now()), LCD_LINE_1)`
`lcd_string("Innovaker 2017", LCD_LINE_2)`
`time.sleep(60)`
	number to be displayed on.

`if __name__ == '__main__':`
`    try:`
`        setup()`
`        main()`
`    finally:`
`        lcd_byte(0x01, LCD_CMD)`
`        GPIO.cleanup()`
	`lcd_byte(0x01, LCD_CMD)`- clears the LCD display screen.

Motion Sensors

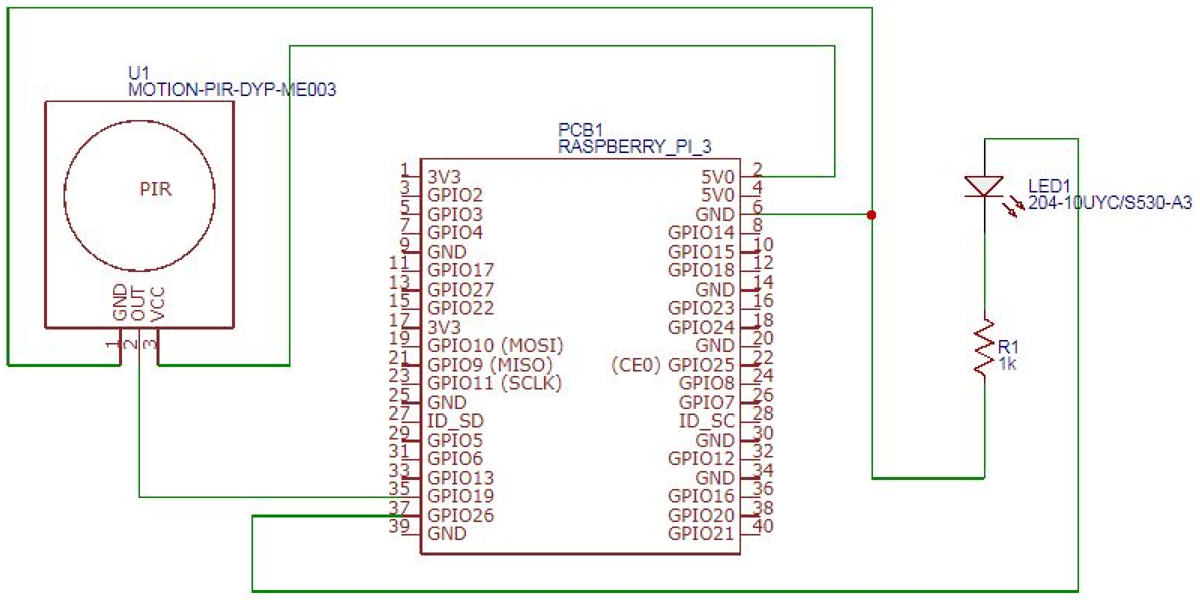
What is a motion sensor?

A motion sensor is a type of electronic sensor that detects for motion or movement. These are often found in security systems or motion sensing lights, but have a multitude of use cases. There are actually two types of motion sensors: active and passive. Active motion sensors emit ultrasonic waves which are reflected back to the sensor. When it detects a large disparity in the echo signal it receives, it sends out a signal to activate a different part of the system. Passive infrared sensors detect infrared or heat signals emitted by humans or other animals. These sensors are typically robust enough to differentiate between changes in daytime temperatures versus that of a living creature. The sensors included in this kit are passive infrared sensors.

How does it work?

The passive infrared sensor is included with 3 pins: one for power, one for ground, and one for the signal. The infrared sensor will detect for any abrupt changes in its infrared signal, and upon detection, will send a digital output to trigger another event.

Building the Circuit



Writing the Code

The following code lights up an LED if motion is detected from the sensor. Much like the button, we can implement this in either a loop or using a callback. These infrared sensors require a buffer time before they can be accessed again, so give them a larger sleep time in between detections.

motion1.py

```
import RPi.GPIO as GPIO
import time

LED_PIN = 10
MOTION_PIN = 18

def setup():
  GPIO.setmode(GPIO.BCM)
  GPIO.setup(LED_PIN, GPIO.OUT)
  GPIO.setup(MOTION_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)
```

The setup for the motion sensor is much like that of a button with an extra parameter for the `pull_up_down`

```
def main():
  while True:
    value = GPIO.input(MOTION_PIN)
    if value:
      GPIO.output(LED_PIN, GPIO.HIGH)
      time.sleep(3
    else:
      GPIO.output(LED_PIN, GPIO.LOW)

if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

We can use the `GPIO.input` function to detect the state of our motion sensor and light up the LED based on that status.

motion2.py

```
import RPi.GPIO as GPIO
import time

LED_PIN = 10
MOTION_PIN = 18

def setup():
  GPIO.setmode(GPIO.BCM)
  GPIO.setup(LED_PIN,GPIO.OUT)
```

Similarly to our callback implementation of the button, we use the `GPIO.add_event_detect` method to add a callback function that is implemented when a motion is detected
The bouncetime parameter offers some buffer

```
  GPIO.setup(MOTION_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)
  GPIO.add_event_detect(MOTION_PIN, GPIO.RISING,
callback=motionCallback, bouncetime=200)
```

time before the motion detection can be retriggered

```
def main():
    while True:
        time.sleep(0.01)
        if GPIO.input(MOTION_PIN):
            print("Motion detected")
        else:
            print('No motion detected')
```

The `main` function simply keeps our program running so our callback can be triggered

```
def motionCallback(channel):
    if GPIO.input(channel):
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(3)
        GPIO.output(LED_PIN, GPIO.LOW)
```

The `motionCallback` function lights up a LED whenever the event is triggered

```
if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        GPIO.cleanup()
```

Temperature Sensors

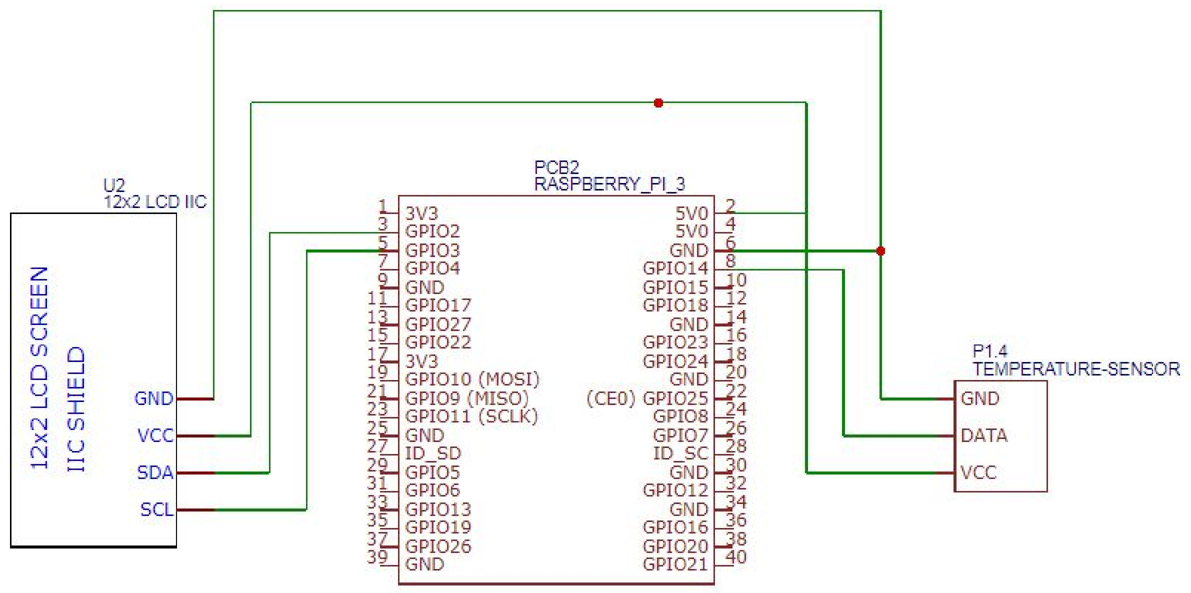
What is a temperature sensor?

There are a number of ways that temperature sensors are implemented, but the basic principle behind all of them is the ability to accurately measure the temperature of the surrounding environment and converts them into a digital signal that can be used. Effectively these temperature sensors act as digital thermometers.

How does it work?

The specific temperature sensor we will be working with is the DHT11, capable of detecting temperature readings between 0 - 50°C and humidity readings between 20% and 80%. The DHT11 has 3 pins attached to it: power, ground, and data out. We will be pairing this sensor with an LCD display to display the information collected

Building the Circuit



Writing the Code

The following program takes in data from the DHT11 temperature sensor and displays it on a 16x2 LCD screen.

temperature.py

```
import smbus
import time
import dht11
import RPi.GPIO as GPIO
from led4import import *

TEMP_SENSOR = 14
```

We have an additional import for the `dht11` sensor in order to use its library

```
def setup():
    GPIO.setmode(GPIO.BCM)
    lcd_init()
```

```
def main():
    sensor_instance = dht11.DHT11(pin =
TEMP_SENSOR)
    while True:
        result = sensor_instance.read()
        if result.is_valid():
            lcd_string("Temperature: " +
str(result.temperature) + "C", LCD_LINE_1)
            lcd_string("Humidity: " +
str(result.humidity) + "%", LCD_LINE_2)
            time.sleep(3)
            lcd_string("Innovaker", LCD_LINE_1)
             lcd_string("Temperature Sensor",
LCD_LINE_2)
            time.sleep(3)
```

In our main function, we use declare `sensor_instance`, an instance of the DHT11 temperature sensor

Using this, we can obtain a reading of the sensor, which we set to the variable `result`. The `result` variable contains data for both the temperature and humidity, which are called upon to be displayed the LCD screen

```
if __name__ == '__main__':
    try:
        setup()
        main()
    finally:
        lcd_byte(0x01, LCD_CMD)
        GPIO.cleanup()
```

Photoresistors

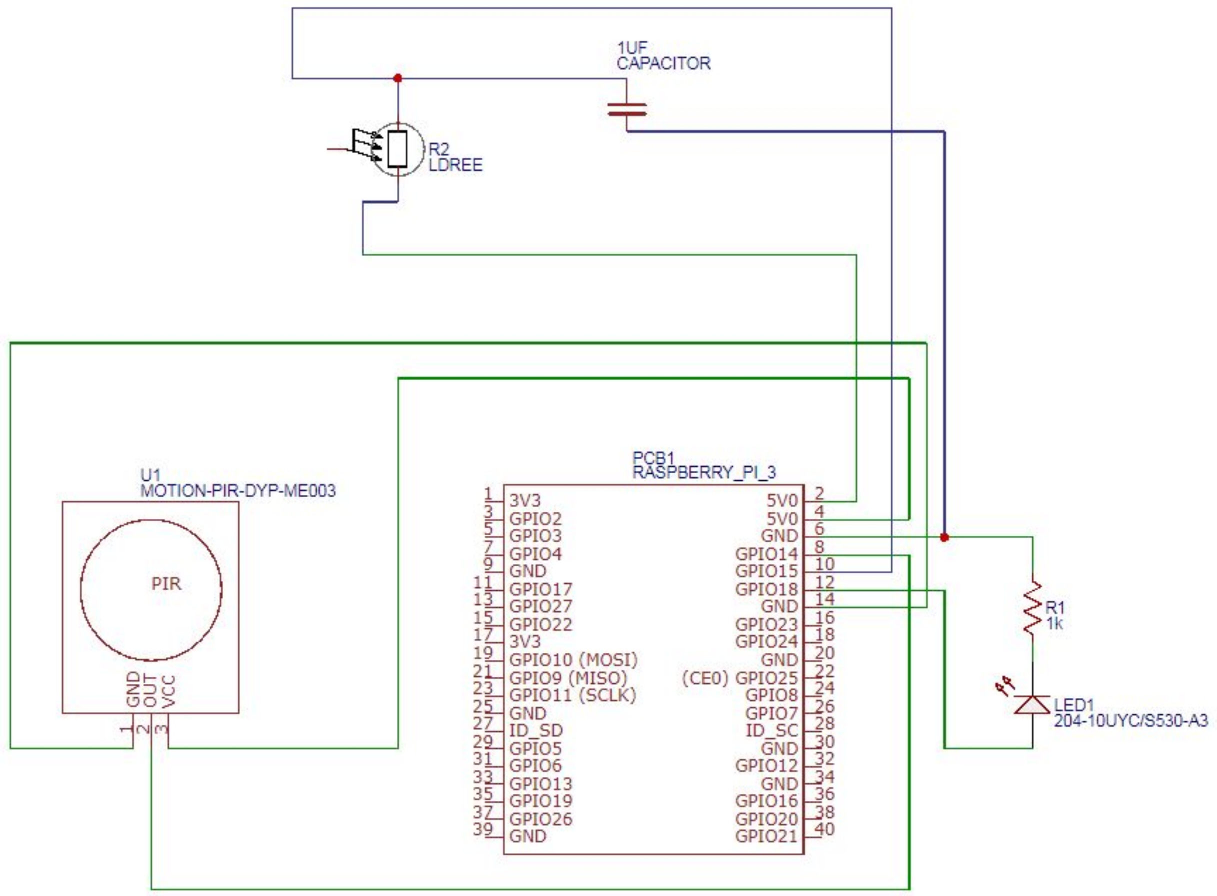
What is a photoresistor?

A photoresistor is a light dependent resistor, meaning the value of its resistance changes according to the amount of light being shed on it. As the light intensity increase, the resistance of the photoresistor decreases. Uses of the photoresistor can be seen in night lights, street lamps, and outdoor clocks. In each case, we can see how the attributes of a photoresistor can be use to create light activated systems.

How does it work?

In order to effectively use the photoresistor with the digital pins of the Raspberry Pi, we're going to need the additional electrical component: the capacitor. Because the photoresistor is simply a resistor with changing values, we are building what's known as a RC (resistor-capacitor) circuit. A special feature about RC circuits is that when we combine the two, it starts to take some time in order to charge up our capacitor. This amount of time is measurable and correlates with the brightness measured by our photoresistor. Thus, the value attained is essentially a measure of the brightness.

Building the Circuit



Writing the Code

The following program uses values from our LDR to obtain a brightness value and light up an LED based on that. Take special note of the `rc_time` method, which we use to attain a brightness value.

LDR.py

```python
import RPi.GPIO as GPIO
import time

LED_PIN = 10
LDR_PIN = 12
```

```python
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_PIN, GPIO.OUT)
```

```python
def rc_time(pin):
```

The `rc_time` method is used to attain a value for

`    count = 0`
 	
`    GPIO.setup(pin, GPIO.OUT)`
`    GPIO.output(pin, GPIO.LOW)`
`    time.sleep(0.1)`
 	
`    GPIO.setup(pin, GPIO.IN)`
 	
`    while(GPIO.input(pin) == GPIO.LOW):`
`        count += 1`
 	
`    return count`
	the brightness. The `count` variable is used as a way to keep track of the time it takes for our capacitor to be charged
	
	First, we setup our GPIO pin as an output pin and allow it to output a low value. Afterwards, we redefine the pin to be a input pin
	
	Within our while loop, we constantly read whether we are still getting a `GPIO.LOW` value. Because charge is slowly being built up in the capacitor, the input from this pin will eventually become `GPIO.HIGH`, causing our while loop to terminate. The amount of time we must wait for this to happen is dependent on the value of the LDR, and thus indicative of the brightness

`def main():`
`    while True:`
`        brightness = rc_time(LDR_PIN)`
`        print(brightness)`
`        if brightness > 600:`
`          GPIO.output(LED_PIN, GPIO.HIGH)`
`          time.sleep(1)`
`        else:`
`          GPIO.output(LED_PIN, GPIO.LOW)`
`          time.sleep(1)`
	Using the `rc_time` method, we obtain real time values for the current brightness. We can then set a threshold which upon being passed causes our LED to light up

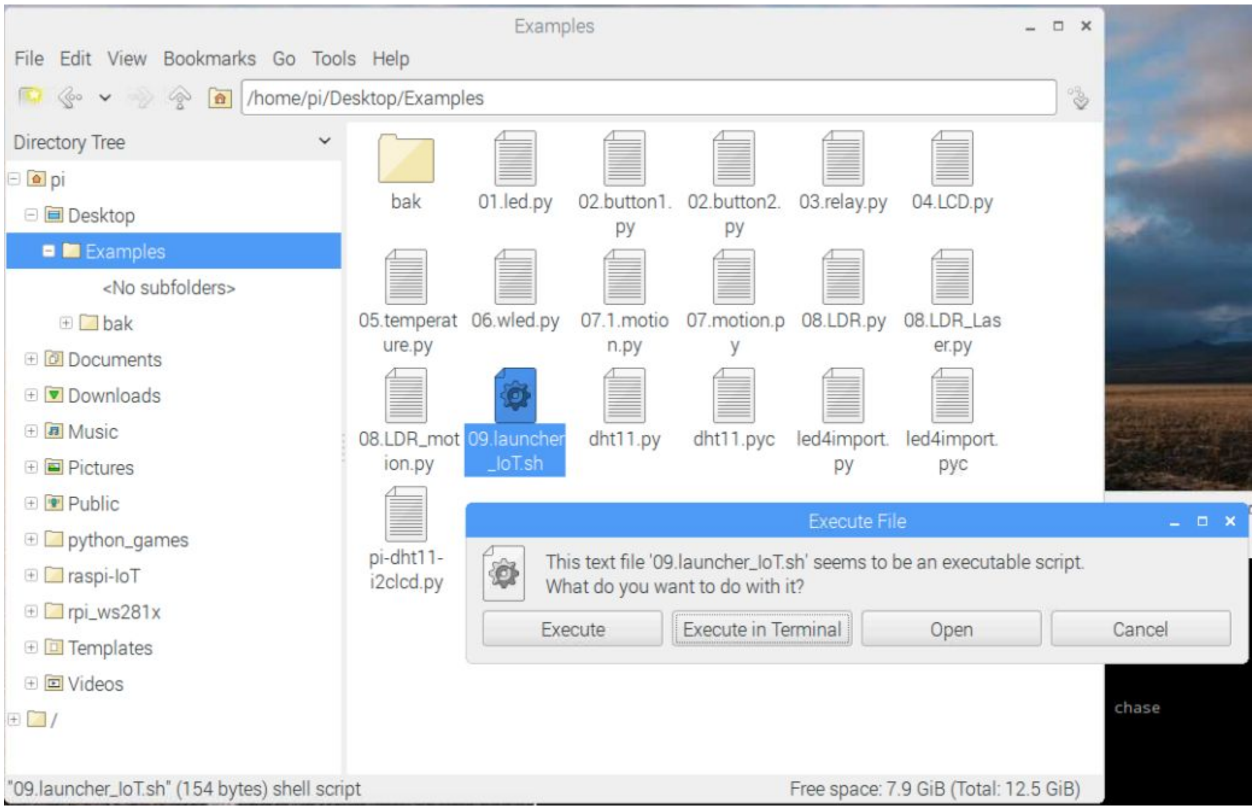
`if __name__ == '__main__':`
`    try:`
`        setup()`
`        main()`
`    finally:`
`        GPIO.cleanup()`	

Projects

IoT

The kits is already set to do IoT, here are some steps and connection.

1. Connect LCD, and LED 25, LED 8, LED 7, LED 16.
2. Double click on “09.launcher_IoT.sh”, a message window display, select “Execute in Terminal”.

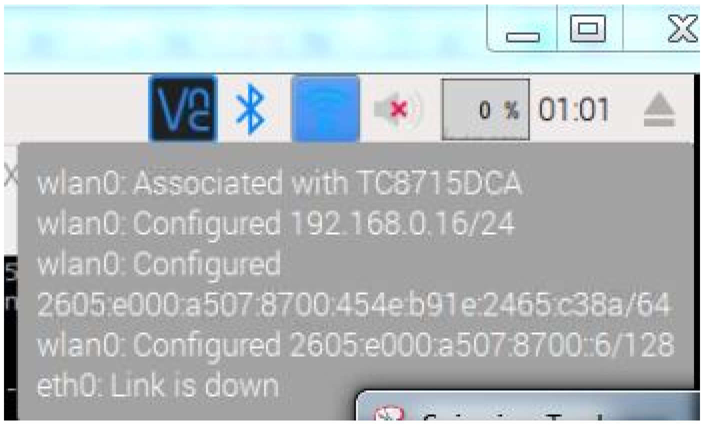


This text file '09.launcher_IoT.sh' seems to be an executable script.
What do you want to do with it?

"09.launcher_IoT.sh" (154 bytes) shell script

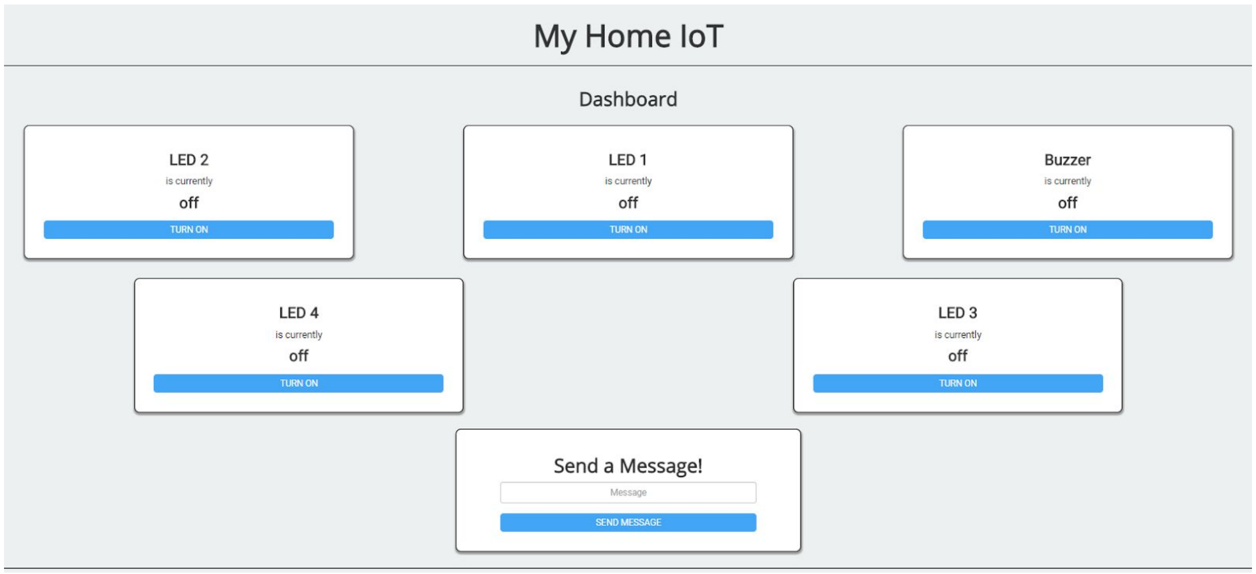
Free space: 7.9 GiB (Total: 12.5 GiB)

3. Find the Pi IP address
Put mouse cursor on wifi, you will find IP address as following: 192.168.0.16



```
wlan0: Associated with TC8715DCA
wlan0: Configured 192.168.0.16/24
wlan0: Configured
2605:e000:a507:8700:454e:b91e:2465:c38a/64
wlan0: Configured 2605:e000:a507:8700::6/128
eth0: Link is down
```

4. On same network computer, in IE or Chrome, type following
192.168.0.16:5000
You will find following UI, it comes from the Pi web server, you can click on buzzer, the buzzer will make sound.
LED 1 = LED 25 on the PCB board.
You can also type message, then click on “Send message” button, the message will be displayed at LCD of the kits.
If you configure your home modem to re-direction web server to above IP address, and set a domain name for your home computer, then others can visit the web from internet, and control your kits.



My Home IoT

Dashboard

LED 2 is currently off

LED 1 is currently off

Buzzer is currently off

LED 4 is currently off

LED 3 is currently off

Traffic Light

Design a circuit that alternates between values of Red, Yellow, and Green, such as those seen in real life traffic lights.

Doorbell

Design a circuit that sounds the buzzer when a button is pressed. See if you can incorporate a responsive light in there as well.

Bonus: Now make your doorbell ring by detecting for motion at your front door! Also take the time to greet your guests with a friendly piece of text on your LCD Display.

Bank Security

Robbers are in the process of putting together a heist to take all the money from your bank! Design a security system incorporating all the components from this kit to alert the police when a robbery is underway to prevent theft.