PROGRAMMING RASPBERRY PI 3 & 4

FOR BEGINNERS

A Step by Step Guide to Write Raspberry Pi Programs and Set up Projects on Raspberry Pi 3 and 4

CHARLES

SMITH

Copyright

Printed in the United States of America

Churchgate Publishing House

USA | UK | Canada

Table of Contents

Why This Guide?

This book is beginners guide to setting up a Raspberry Pi from scratch, how to write codes for your Raspberry Pi, build circuits to connect your Raspberry Pi to the outside world and carrying out complex tasks like automation and interacting with the Internet of Things (IoT).

We will also look at a diverse range of necessary skills to make this journey worthwhile, like computing, electronics, and programming. We will carry out most of our projects with the Raspberry Pi 3 Model B+ and the New Raspberry Pi 4.

About the Author

Charles Smith is a tech enthusiast with over 13 years of experience in the ICT industry. He is a geek, and passionately follows the latest technical and technological trends. His strength lies in figuring out the solution to complex tech problems. Charles holds a Bachelor and a Master's Degree in Computer Science and Information Communication Technology respectively from the MIT, Boston, Massachusetts.

Chapter One

Introduction

What is a Raspberry Pi?

In General, a Raspberry Pi is a single board computer the size of a credit card and requires an operating system to run. The Raspberry Pi has three main purposes – because it is a low power consumption unit, it can be used in different situations where a personal computer will not be usable, e.g. robotics, a server, webcams, etc. The second place where the Raspberry Pi is useful is hobbyist programming (e.g., Python, C++, Java, Rasp, Linux, etc.) and thirdly, it is an interfaced between software and hardware. Hardware is attached to the pi via the General Purpose Input-Output Pins (GPIO). You can have temperature monitor, humidity monitor, switches, LEDs, etc connected to the pi for all kind of projects whether ecological, weather, robotics, drones, motion detection, etc. Because of its versatility, low-power need and low cost; Raspberry Pi is attractive to the hobbyist, embedded system programmers, and students. There are plenty of operating system options for Raspberry Pi, and any of them can get the job done. Examples of Raspberry Pi include Raspbian, Windows 10 IoT core, Retropie and Openelec.

1. Raspbian: This is the most popular, stable, and general purpose Raspberry Pi Operating System.

2. Windows 10 IoT Core: This is a striped-down version of the Windows 10 Operating System, developed for the Internet of Things (IoT).

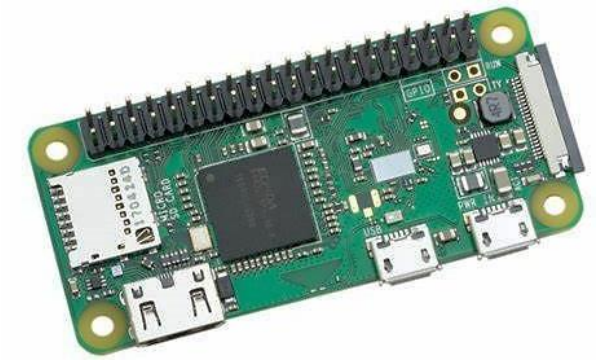
3. Retropie: Retropie is an operating system dedicated to turning the Raspberry Pi into a gaming console.

4. Openelec: Openelec is used to turn the Raspberry Pi into a media server to surf local or online media on a television using the Raspberry Pi as a media server.

Evolution of the Raspberry Pi

There are different Raspberry Pi models out there in the market. Some of them include – Raspberry Pi 4, Raspberry Pi Model 3, Raspberry Pi 2 Model B, Raspberry Pi Model A+, Raspberry Pi Zero W (wireless), Raspberry Pi Compute Module 3 etc. We will now briefly look at some of them.

Raspberry Pi Zero W (wireless)

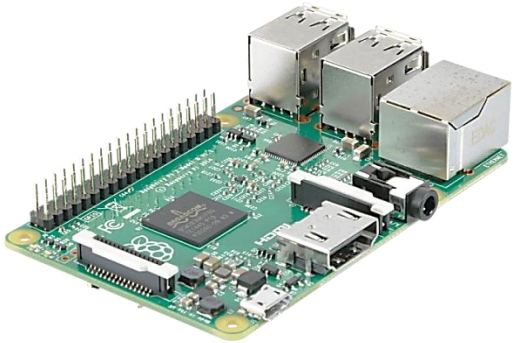


Raspberry Pi Zero is a striped-down miniature version of the Raspberry Pi. However, it does have onboard Wi-Fi and Bluetooth. The Raspberry Pi Zero is ideal for miniature projects.

Raspberry Pi Model A+

The Model A+ is a basic model of the Raspberry Pi. It consists of a fewer USB port, absence of Wi-Fi and Bluetooth connectivity and half the processing power of the Raspberry model 3 (700MHz).

Raspberry Pi 2 Model B

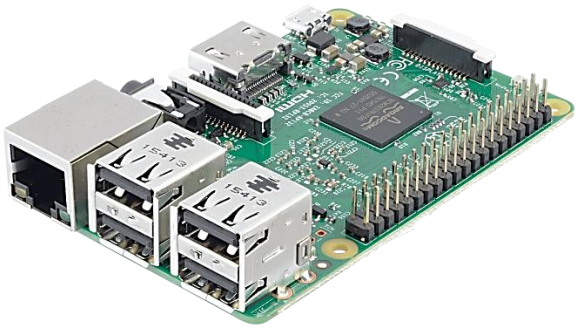


This model of Raspberry is around for legacy reasons; it has a slightly slower processor 900MHz and 1GB of RAM. It does not come with Wi-Fi and Bluetooth.

Raspberry Pi Compute Module 3

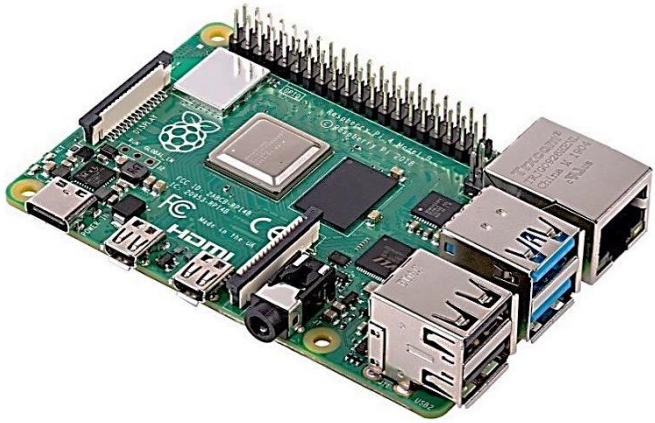
They are used for more engineered solutions. Rather than having all the onboard hardware peripheries, it breaks out the pins of the processor into RAM-like card-edge connectors so that engineers can create custom boards and use only the peripheries that they need. It comes with a 1.2GHz Quad-core processor and 1GB of RAM.

Raspberry Pi Model 3



Raspberry Pi Model 3 is amongst the latest Raspberry Pi models. It has the interface you would find on a full-sized computer like the HDMI port, audio-out, Ethernet, USB ports, general-purpose input, and output pins, Bluetooth and Wi-Fi connectivity. Raspberry Pi model 3 comes with a 1.2GHz processor and 1GB RAM.

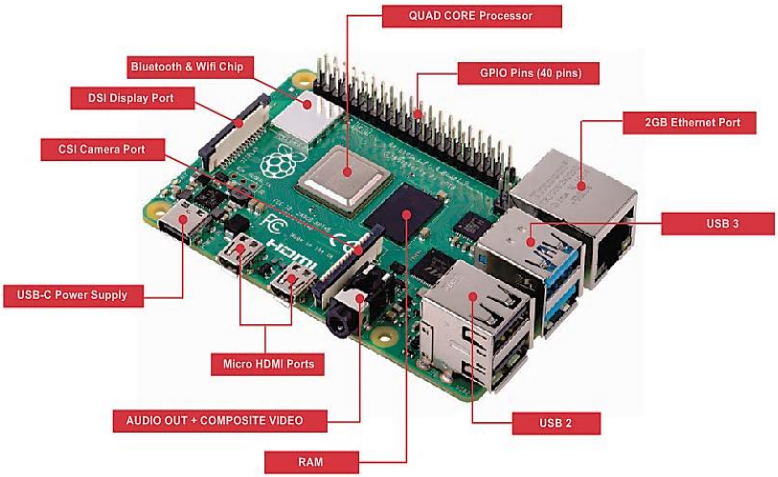
Raspberry Pi Model 4



The Raspberry Pi 4 is a big upgrade compared to the Pi 3 and the 3 +. The CPU got a major upgrade in terms of performance. The Pi4 came with a 1.5GHz Quad-Core ARM Cortex-A72 which is different from the Quad-Core ARM Cortex-A53 found on the Raspberry Pi 3 and 3+. The A72 setup is three times faster than the A53 setup. The Raspberry 4 also got an upgrade from USB-2 to USB-3, and the band of the internal buses has been upgraded to cope with data transfer speed. Raspberry 4 has 2 Gigabit Ethernet which is

a shift from the 1 Gigabit obtainable in the Pi 3 and 3+. Like the Pi 3, there is an onboard wireless network connectivity and Bluetooth 5.0 on the Raspberry Pi 4. Up until the introduction of the Raspberry Pi 4, we only had between 500MB – 1GB of RAM, which was the maximum, but now we have 1GB, 2GB and 4GB options. There are other noteworthy significant changes on the Raspberry Pi 4 from the Pi 3. First, there are no longer full size HDMI ports onboard the Pi 4; we now have two micro-HDMI ports that support 4k resolution display. This makes it possible to connect two monitors to the Raspberry Pi 4 simultaneously. The second change is that we have moved from using micro-USB to power the board to USB-C. Other than these changes, the Raspberry Pi 4 follows the same model and styling of the Raspberry Pi 2 and 3 before it. You've got, for example, that connector where you can connect the Raspberry Pi's camera. Audio jack, slot for Micro-SD card, etc. The Raspberry Pi 4 is compatible with everything that goes on with the Raspberry Pi 3. However, a downside of the Raspberry Pi 4 is that it overheats when pushed to its limit. This, however, doesn't interfere with the function, but it brings down the clock speed significantly while it waits for the chip to cool down. A thermometer icon is displayed on the screen to indicate overheating.

Raspberry Pi components

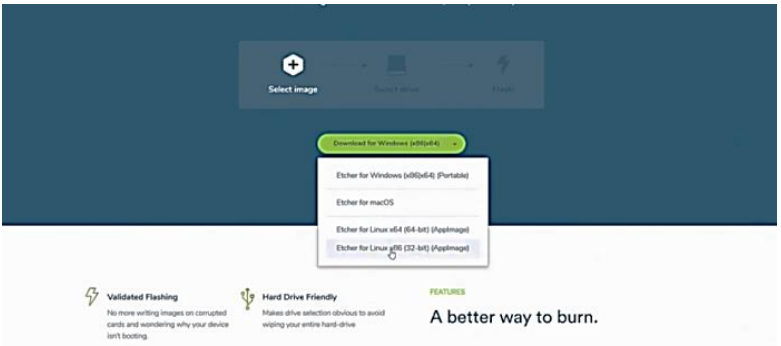


Installing Raspbian on Raspberry Pi 3 and 4

The installation process for both Raspberry Pi 3 and 4 are the same. To get the Raspberry Pi up and running, you would need to download a package called New Out of the Box Software (NOOBS). NOOBS is an operating system installer package that has Raspbian but also allows users to run a few other Raspberry Pi operating systems which can be downloaded from the internet. To Install NOOBS;

- Visit www.raspberrypi.org
- Click on "Downloads" on the dashboard
- Search and download the full installation "NOOBS" or download the Raspbian OS directly. If you are downloading the Raspbian OS directly, go with the "Raspbian Buster with desktop and recommended software" in a zip folder. This version has a lot of

things preinstalled, so you don't have to do any setup. While the Raspbian is downloading, you will need an application called "Etcher" to flash the Raspbian image to the SD card so you can run it on the Pi. Go to www.balenaetcher.io/etcher. Download either the Windows, Linux or Mac version depending on your PC.

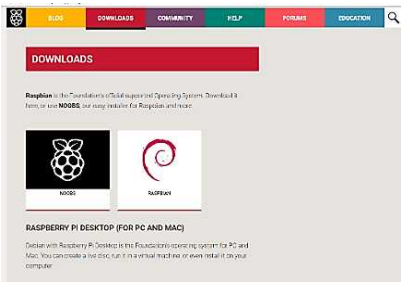


Copy the etcher and Rasbian zip folder to your desktop.

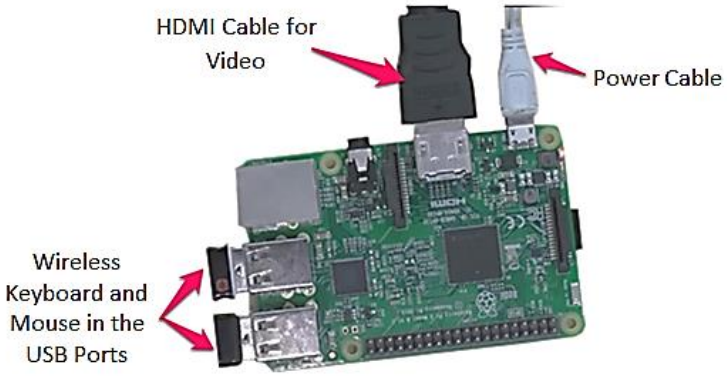
Next, click on the belanaEtcher software to launch it. Select the image you want to flash to the SD card (i.e., the Rasbian

zip folder). Make sure you are flashing to the correct SD Card. Once you are sure of the SD card, click "Continue" and then "Flash." Etcher will go ahead and install the operating system on the SD card. When Etcher is done flashing Raspbian on your SD card, you will get some warnings, close the 'warning' dialogue box. Take the SD Card out of your PC and insert it in your Raspberry Pi 3 or 4. Connect the Raspberry to a monitor(s) and then connect it to a power source. If everything is working properly, we will get a solid red light and a flashing green light. The flashing green light tells us the Raspberry Pi is reading from the SD card. On the first boot, Raspbian will automatically expand the file system on the SD card so we can use the full capacity. It automatically reboots, bringing us to the Raspbian Desktop with the setup wizard. Click 'Next' to choose country, language, time zone and select keyboard type. Click the checkboxes for "Use English language" and "Use US keyboard" if you live in the United States. The keys on the European version are different from the one on the US keyboard. Next, set your password; the default password is 'raspberry.' Go ahead and put your new password, confirm it and click 'Next'. Select your wireless network. You can skip this part if you plug the Ethernet cable into the Pi. Next, you will be asked to 'Update software.' You might get an error message here; we will look at how to update our software manually later on. Finally, click on 'Restart,' and it will take you to the Raspbian desktop.

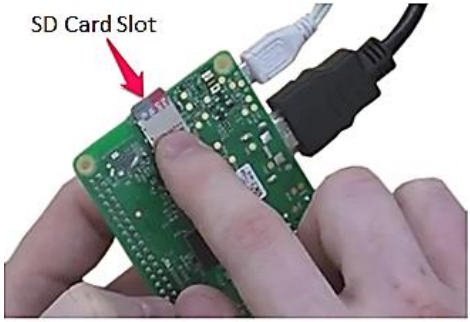
On the other hand, if you prefer to install Raspbian using NOOBS; download the NOOBS setup file on your PC.



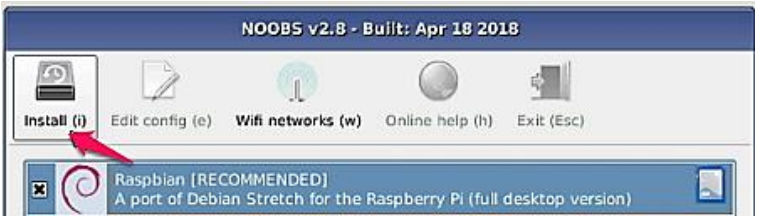
- Plug in an SD card of minimum 8GB into your PC.
- Extract the NOOBS zip file and transfer the extracted setup folder to the SD card connected to your PC.
- Make the necessary cable and wireless connections to your Raspberry Pi 3 or 4.



- Insert the SD card with the NOOBS setup folder into the SD card slot at the back of the Raspberry Pi.

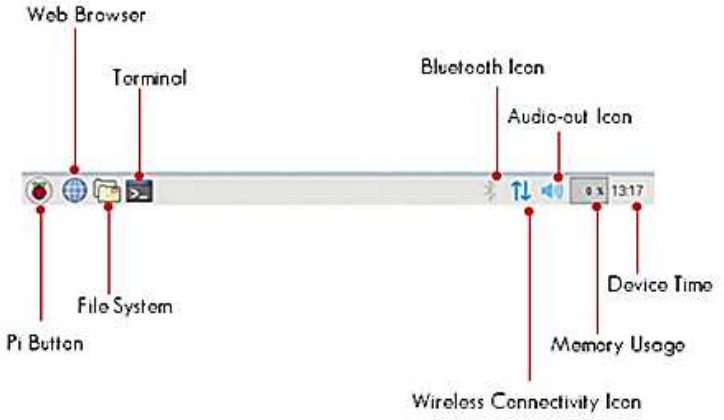


- Boot up the Raspberry Pi.
- After plugging the power and connecting the Raspberry Pi to a monitor, you will be presented with the NOOBS boot menu.
- Select 'Raspbian' and click on 'Install'



- After installing 'Raspbian,' you will be presented with a welcome page and then the Home Screen.
- Follow the setup process to get the Raspberry Pi working.

Raspberry Pi Applications Bar



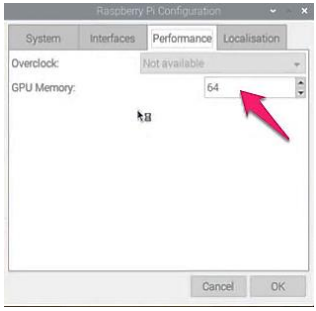
At the top is the Applications bar; this holds the pi button, web browser, file system, terminal, Bluetooth icon, wireless connectivity icon, audio out icon, memory usage, and device time. Beneath the application bar is the desktop. The Pi button is like the start button on a Windows Operating System. It opens the Pi menu when clicked. The Pi menu includes

programming, Education, Office, Internet, Sound & Video, Graphics, Games, Accessories, Help, Preferences, Run, and Shutdown Menus. Each of these menu items has sub-menus.

Setting up the Home Screen

The first thing you do after setting up your Raspberry Pi, is to increase the GPU memory to at least 512MB. This is shared with the built-in RAM. To do this, click on the Raspberry icon – preferences – Raspberry Pi configuration – performance.

Under "performance', increase the GPU memory to 512MB



Click "Ok". However, this setting will not take effect until you restart the Raspberry Pi.

The next thing you need to do after increasing the GPU is to get rid of the underscan lines. The underscan lines are the black bars around the edges of the screen. To do this, 'Click' on the Pi button – preferences – Raspberry Pi configuration. On the dialogue box that pops up, click the radio button to DISABLE "Underscan" and click on "OK." This will require a reboot. Click on 'Yes', and when the Pi is done rebooting, the desktop would fill up the entire screen.

If you did not setup keyboard preference during setup, you could do that by clicking on pi button - go to Preference – Mouse and Keyboard settings – on the dialogue box that comes up on the screen, click "Keyboard Layout" – select "United States" on the list of countries displayed at the left-hand side, then select "English US" and "Ok."

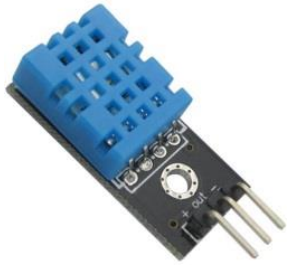
Raspberry Pi Sensors and Module Kits

Most basic Raspberry pi projects require the following;

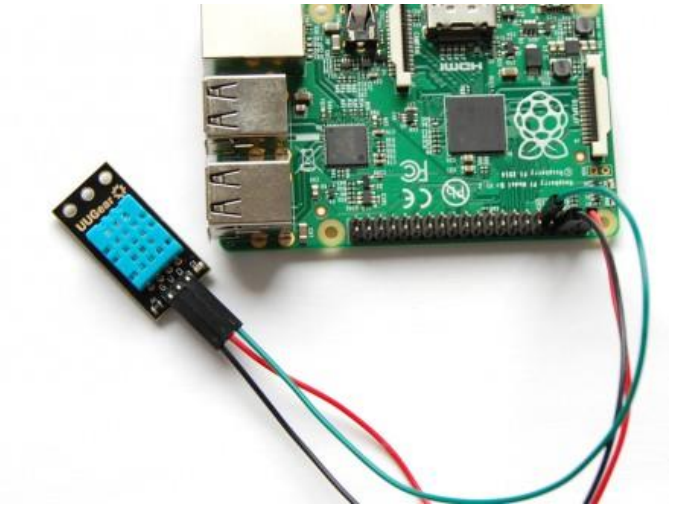
1. A Raspberry Pi
2. A breadboard
3. Jumper wires
4. Resistors and
5. Sensors
6. Modules

Sensors are essential components for most automated Raspberry Pi Projects. However, there are several Raspberry Pi sensors for different projects, and they come in module kits. A typical kit contains between twenty-seven to forty-seven sensors and electrical chips depending on the kit you purchase. We will go through a few sensors and chips you would find in Raspberry sensor kit and some of the projects they are ideal for.

Temperature and Humidity sensor



This sensor is used to measure the amount of water vapor in the air. It is required in projects that would detect the likelihood of precipitation, dew or fog. It has components that measure resistive-type humidity, an NTC temperature and relative humidity. A Humidity sensor is a requirement when building a weather station using the Raspberry Pi. The Blue box on the chip is a THT 11 sensor. The Sensor has three pins. A ground Pin (GND), +5v pin and a signal pin.



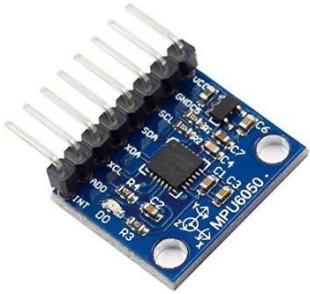
Human Touch Sensor

This sensor detects electrical conductivity from the body. It is used in projects that requires a human touch to start. It is also very useful for machinery safety.

Percussion Knock Sensor

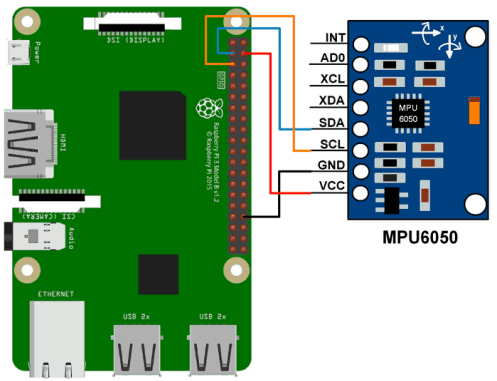
This sensor detects sound. It can be connected to an LED on the Raspberry Pi. The LED is lit when there is a knock or a sound. It is also used in projects that require sound to trigger an event.

MPU 6050



This is a micro electro mechanical system. This sensor comes with a free access gyroscope, accelerometer, a temperature sensor, and a digital motion processor. An accelerometer measures non-gravitational acceleration while the gyroscope

uses earth's gravity to determine orientation. The digital motion processor can be used to process complex algorithms directly on the Raspberry pi board. It processes algorithms which turns raw values from the sensors into stable precision data. On the Sensor, we have the VCC for voltage supply (can be connected to 3.3v and 5v sources), Ground (GND), Serial Clock line (SCL) and Serial Data Line (SDA) – Both the SCL and SDL pins are part of the I2C serial bus which requires only two wires for communicating, Auxiliary data pin (XDA) which is an I2C master serial Data pin, Auxiliary clock (XCL) which is an I2C master serial clock pin. Both the XDA and XCL can be used to connect to multiple external sensors with the I2C bus. The XDA allows the chip to gather a set of data from other sensors without intervention from the central system processor. The last two pins are the AD0 pin and the INT pin. INT stands for interrupt. The MPU6050 is used in several Raspberry Pi projects including robotics and drones.



Laser head Sensor

This is a sensor is mostly used in motion detection projects. It transmits a laser beam from the laser head to a laser receiver. If anything comes in between the laser head and the receiver, it breaks the laser ray and triggers a buzzer. It is ideal to add a buzzer to a motion detection project to set off an alarm when there is an intruder.

A Passive Buzzer

DS1302 Clock Module

This module is used for projects that require correct date and time displayed even when the Raspberry Pi is switched

off. The clock module has an onboard battery that keeps the system running. This module works with an LCD to display time and date.

Soil Humidity Sensor

This sensor measures soil humidity. It is used mostly in projects that require testing the amount of water in the soil. For example, a Raspberry Pi can be connected to a sprinkler which is automatically switched ON when the moisture in the soil is low. This is useful in Lawns, gardens, football pitch, etc.

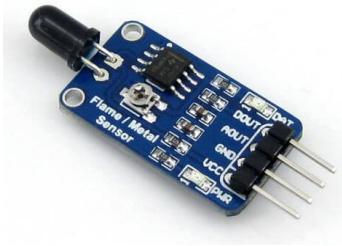
Other important sensors and module you will come across as you move from basic to advanced Raspberry Pi projects include;

- Microphone sound sensor

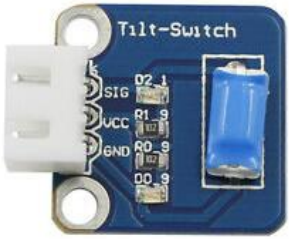
Optical breaking sensor

Hall Magnetic Sensor

Flame sensor

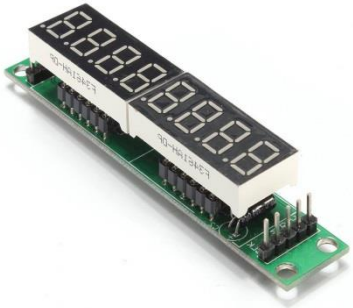


Tilt switch



Infra red emission sensor

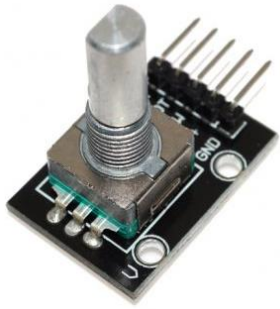
7 or 8 Segment Display



5V Relay

Magic Light Cup

Rotary Encoder



Bread board Power Supply

Photo resistor

SD Card Reader

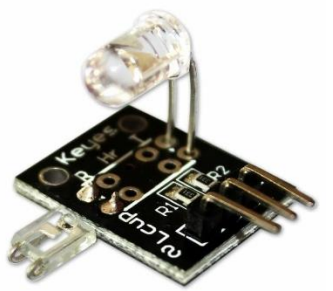
\

Smart Car Obstacle

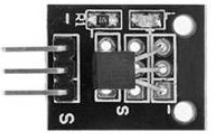
Sensor infrared

PS2 Joystick Game Controller

Finger Detect Heartbeat sensor



Temperature Sensor (digital)



Water Level Sensor

Adjustable step down

Vibration Sensor

Gas Sensor

Carbon monoxide Sensor

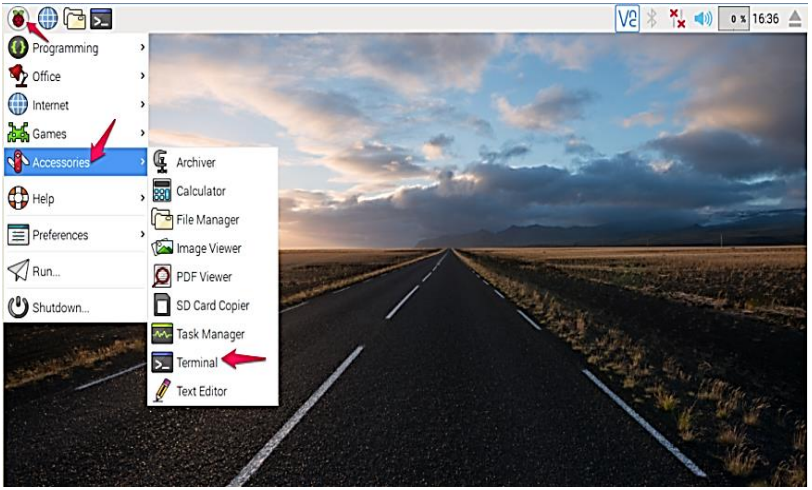
HDMI LCD

Chapter Two

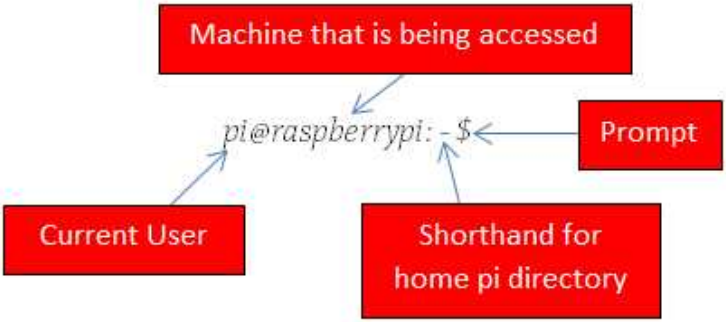
Making Files and Navigating the File System

Introduction to the Shell

The shell is a powerful Raspbian tool when it comes to writing programs. It is a program that processes commands entered and returns outputs. There are two ways to access the shell. First, we can run the tunnel on the Raspbian taskbar. The other method is to click on the Pi button to open up the Pi menu – click on Accessories – Terminal.

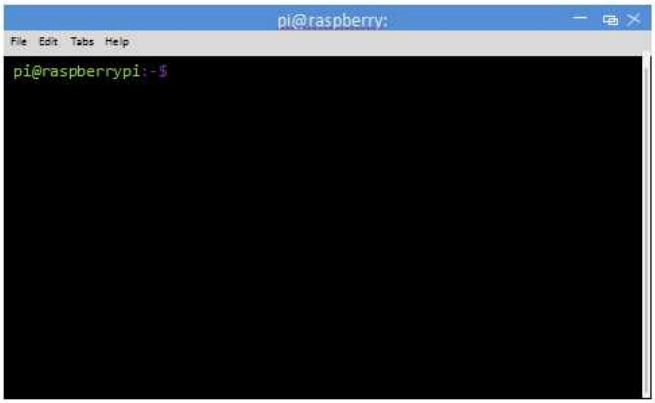


When you access the terminal, what pops up on the screen is a prompt. The prompt is a shell waiting for you to input a command.



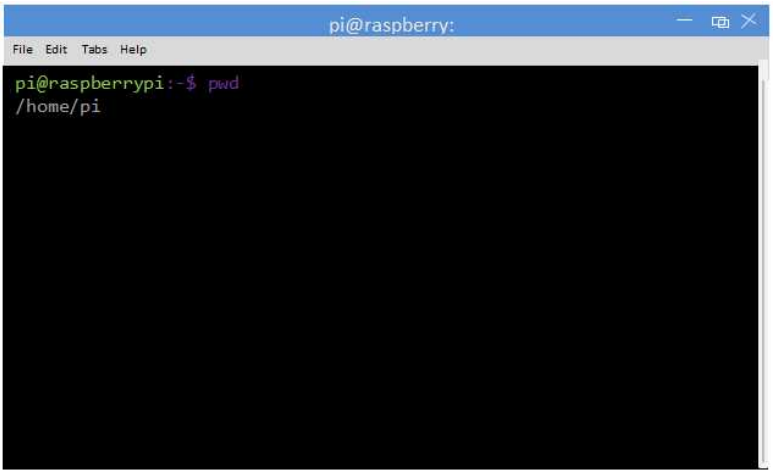
where:

{pi}	*is the current user*
{raspberry:}	*is the machine that is being accessed*
{~}	*is the Home Pi Directory*
{$}	*is the prompt. It is the shell waiting for our command*



The above shell is waiting for us to input our commands. For instance, if we type in "print working directory," which is represented by "pwd" after the shell, it tells us in what directory we are. For instance, when we input the "pwd" command, it indicated we are at the "home directory" as shown below;

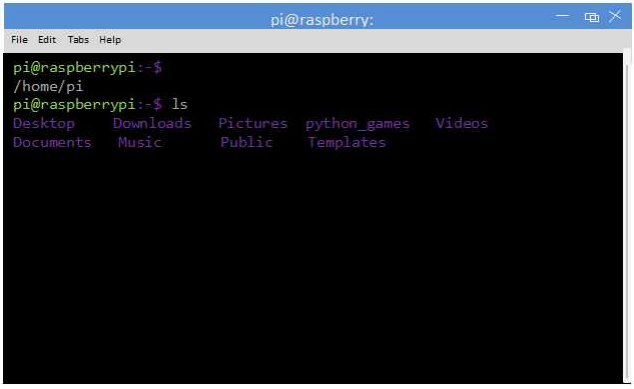
/home/pi



We can as well list the contents of the directory we are currently in using the {LS} command as shown below;

pi@raspberrypi:~ $ ls

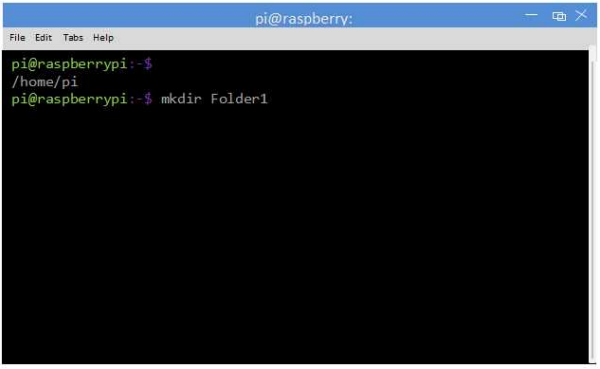
This will list out all the directories (e.g., Desktop, Downloads, Pictures, python_games, Videos, Documents, Music, Public, Template, etc and we can also access any of the listed directories.



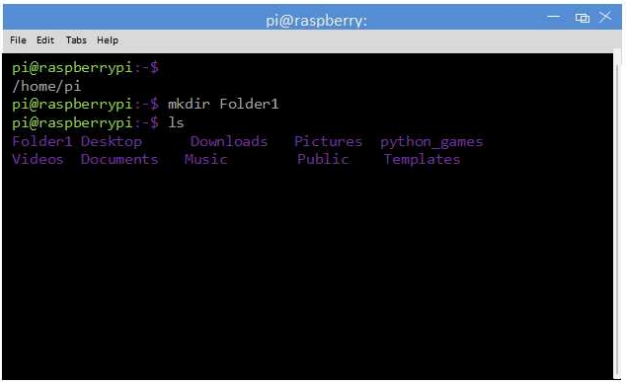
We can alsocreate a new directory using the "make directory command," which is 'mkdir'. For instance, let's create a directory called folder1; the command to do this is shown below.

pi@raspberrypi: - $ mkdir folder1

This will create a new directory called folder1.



If we execute the {LS} command again, a new directory (folder1) will be added to the list of directories this is displayed as shown below;

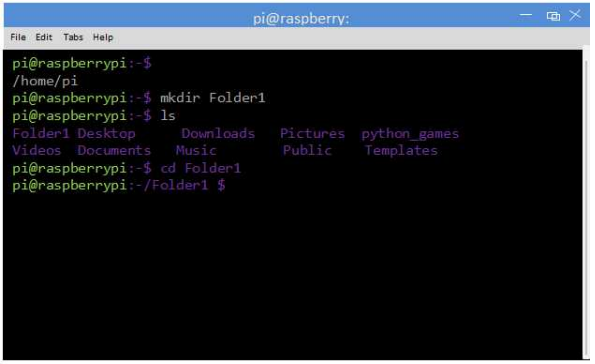


You can also access any of the directories using the {cd} command. For instance, if we wish to access the newly created directory called folder1, we will use the command below;

pi@raspberrypi: - $ cd folder1

This will change our prompt to

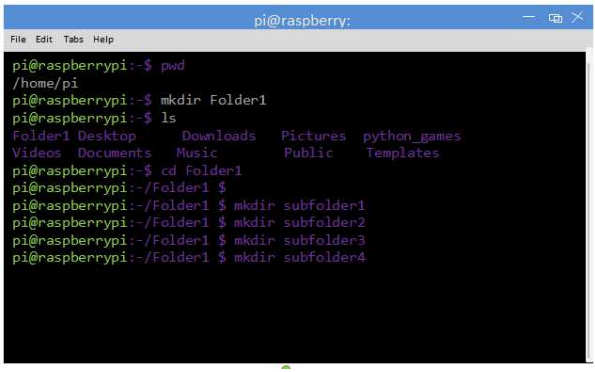
pi@raspberrypi: -/folder1 $



We can also create sub-directories within a directory. For instance, we can create four different directories within

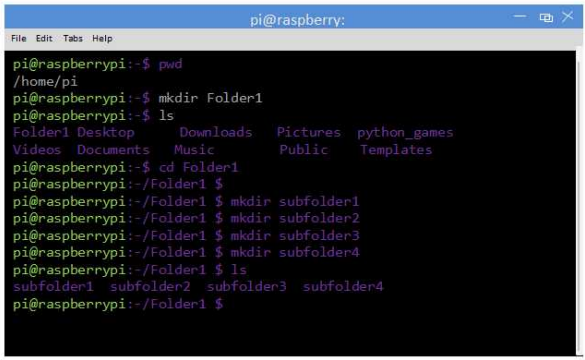
"Folder1".Let's call the four directories subfolder1, subfolder2, subfolder3 and subfolder4. We write the command thus;

pi@raspberrypi: -/folder1 $ mkdir subfolder1
pi@raspberrypi: -/folder1 $ mkdir subfolder2
pi@raspberrypi: -/folder1 $ mkdir subfolder3
pi@raspberrypi: -/folder1 $ mkdir subfolder4



If we execute 'LS' command, the new subdirectories will be displayed thus;

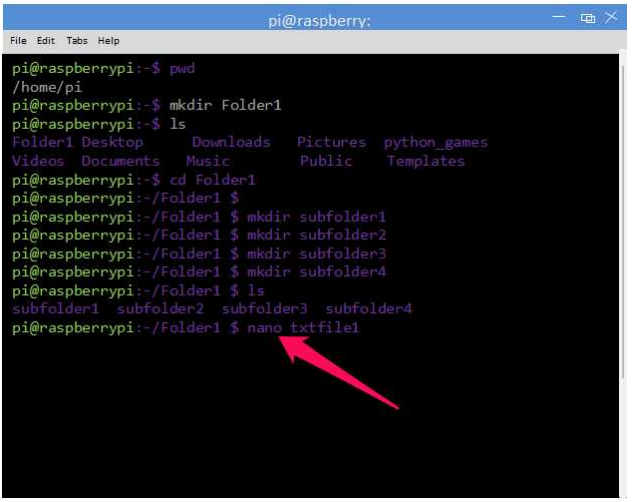
subfolder1 subfolder2 subfolder3 subfolder4



Now that we have created folders and subfolders, let's create a text file. Text files can be created using the "nano text

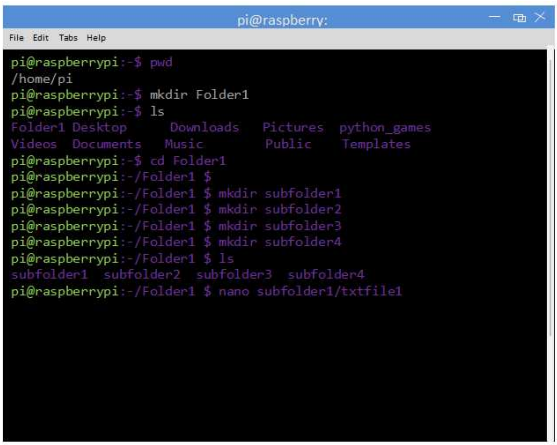
editor." Executing "nano" opens the nano program where we can write our file. If you already know what you want to name the file, lets say we want to name it "txtfile1," then we just enter the command below in the directory we are currently in:

pi@raspberrypi:-/Folder1 $ nano txtfile1

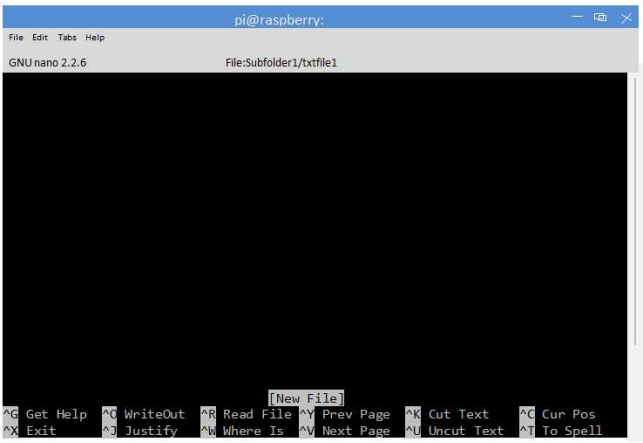


There are two ways you could create a text file; firstly you can navigate into the directory or subdirectory and create the text file using the "nano" command or while creating the text file, you could enter a relative pathwhere you wish to put the text file. For instance,

pi@raspberrypi:-/Folder1 $ nano subfolder1/txtfile1

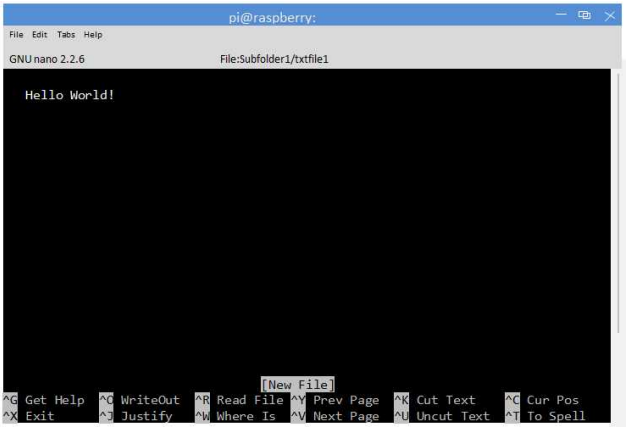


This command means, from where I am currently, there is already a directory called "subfolder1". In that directory, I want to create a text file. When you press the "Enter" key on the keyboard, you will leave the shell, and you be presented with the nano editor.

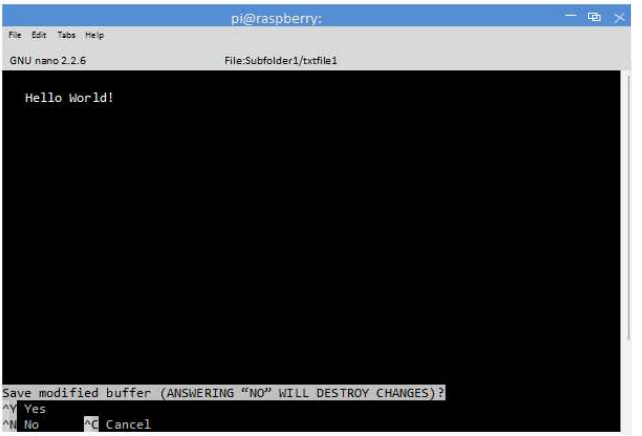


The nano editor screen has a provision for inputting text. At the bottom is the Menu. The "^" represents control "ctrl" on the keyboard. For instance "Ctrl X" will exit the program and

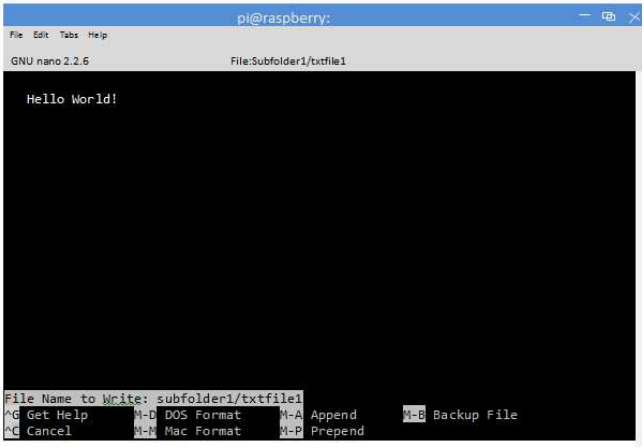
"Ctrl G" opens up the Help Menu. After typing your text, you can exit the nano editor, after saving your text and back to the shell. For example, we will write the text "Hello World!" in our text file.



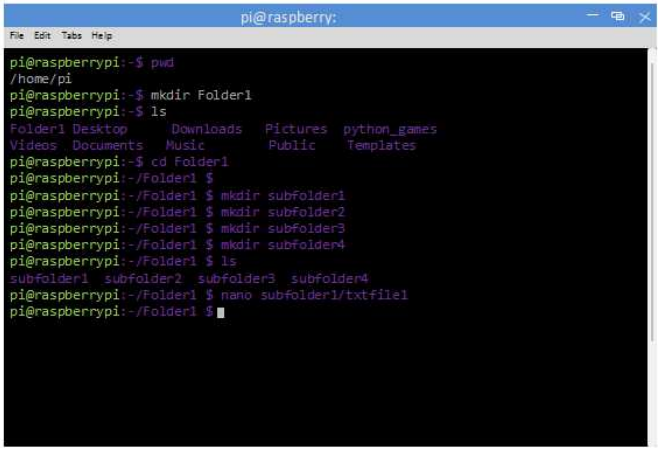
Click on "Ctrl X" to Exit the program. You will get a message such as "Do you want to save the changes?" Select "Y" for "Yes."



Because we've already entered a file name for the above text file, it already populated subfolder1/txtfile1into the "file name to write" as shown below.

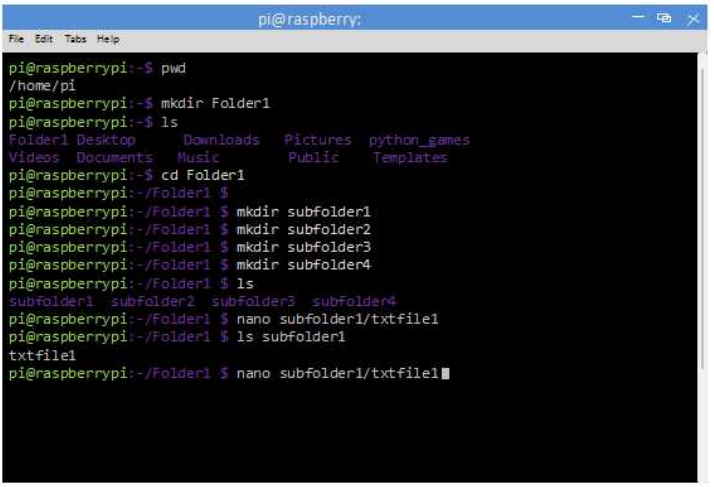


Press the "Enter" key on the keyboard to exit nano mode and go back to the "shell."



Rather than jump into the "subfolder1" files with the "cd command" and executing LS command to see if our text file is still

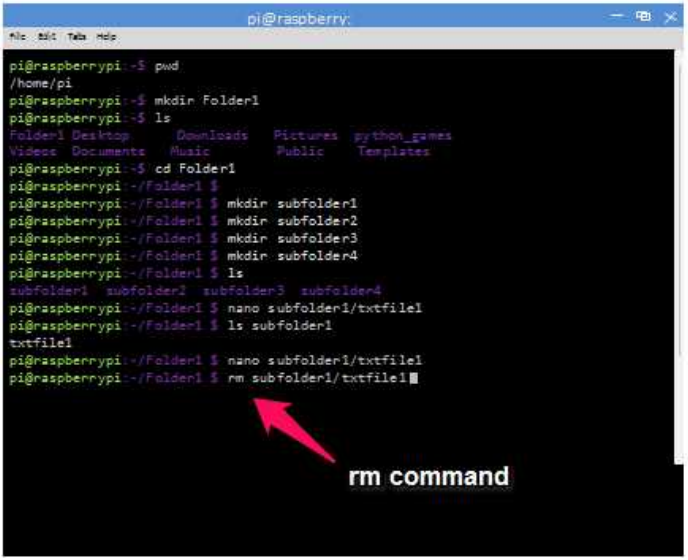
there, we can just say; *pi@raspberrypi:-/Folder1 $ nano txtfile1 ls subfolder1* to access, edit or modify the text file. We don't need to be in the subfolder1 directory to execute this. All you need do is to execute the command *nano subfolder1/txtfile1* and press the "Enter" key on the keyboard to take you to the stored text file as shown below;



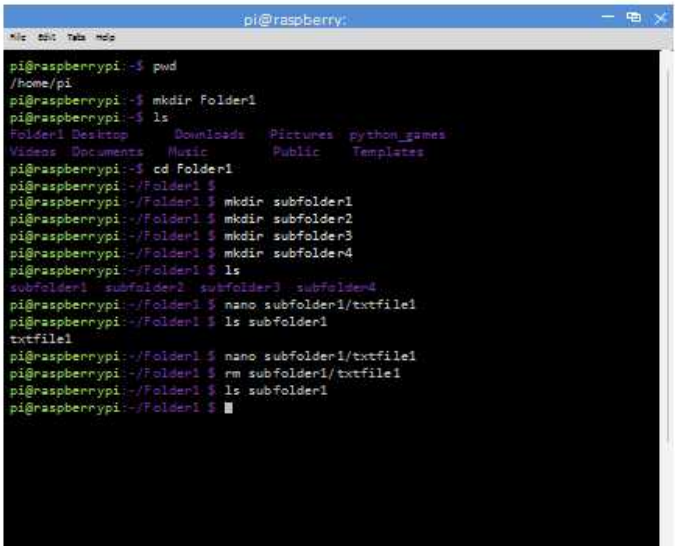
We can delete a text file without going into the directory or subdirectory to carry out this action. We can execute this action using the "rm command." For instance, if you wish to delete the text file we named "txtfile1" which was saved in subfolder1 directory, without going into the directory, execute the "rm command."

pi@raspberrypi:-/Folder1 $ rm subfolder1/txtfile1

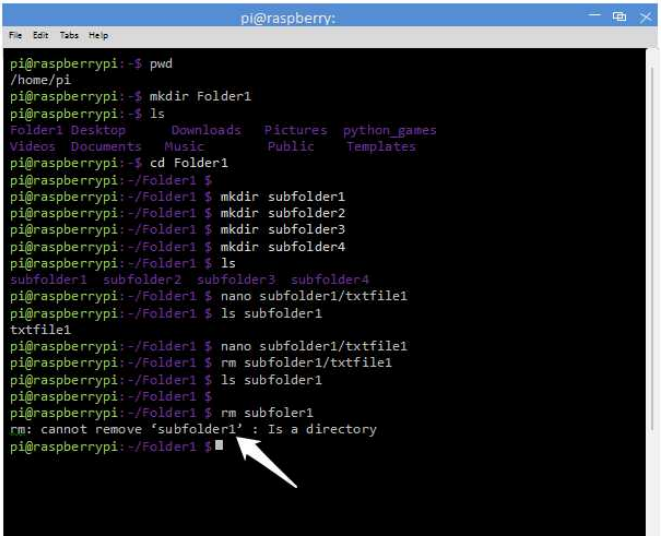
When this command is executed, there is usually no feedback, but to be sure the command has been executed, perform "ls command" for subfolder1, and it will return empty.



To be sure the deletion of the text file is successful, perform an "LS command." It won't return the text file named "txtfile1" as shown below.

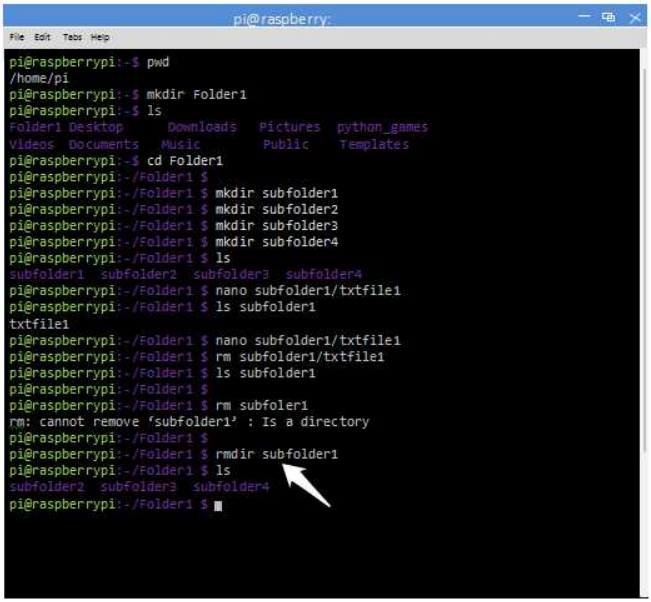


So far, we have managed to make directories, write a text file, and delete it. Now, let's look at how to delete a directory. Let's see if we can use the 'rm' command to delete a directory. Let's try to delete the subfolder1 directory using 'rm' command



You can see that when we tried tovdelete the 'subfolder1' directory, it returned a message saying 'rm cannot remove 'subfolder1 because it is a directory. This means the rm command can be used only to delete text files and not directories.

There are several commands to remove directory, and this depends on if the directory has content or empty. To delete an empty directory, we make use of the command "redir." If you try to use the "redir command" for a directory that has a text file, it will not work. Now let's remove the subfolder1 directory and use the LS command to see if it worked.

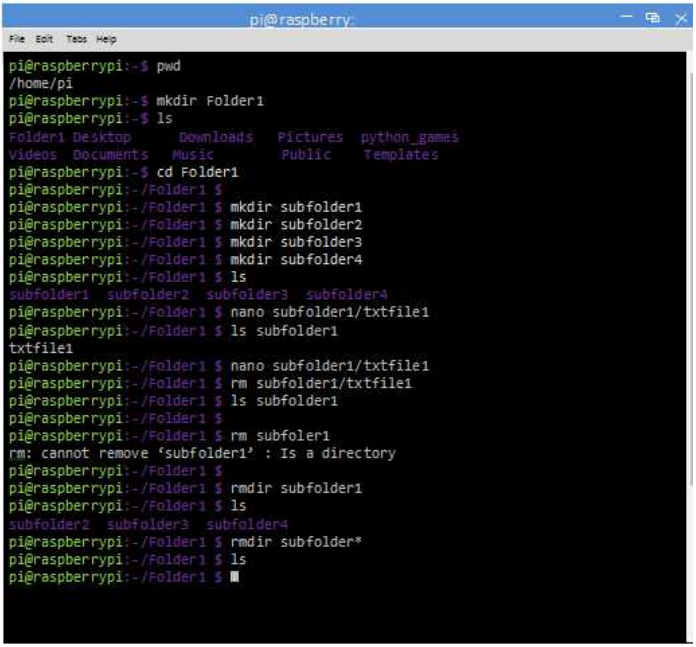


Note that after executing the rmdir on subfolder1, only subfolder2, subfolder3, and subfolder4 directories were left. When we executed the LS command, it showed that subfolder1 directory was successfully removed. We can remove multiple directories or subdirectories using the command

*pi@raspberrypi:~/Folder1 $ rmdir subfolder**

*where asterisks * stands for any number of characters*

The above command, when executed removes the entire subfolders directories in Folder1. We do this, instead of writing *pi@raspberrypi:~/Folder1 $ rmdir subfolder2 subfolder3 subfolder4*. LS command would return nothing because Folder1 no longer has subdirectories.



If the screen is too cluttered or messy, you can type in "clear" and this will clear the window.

Updating Raspberry pi OS

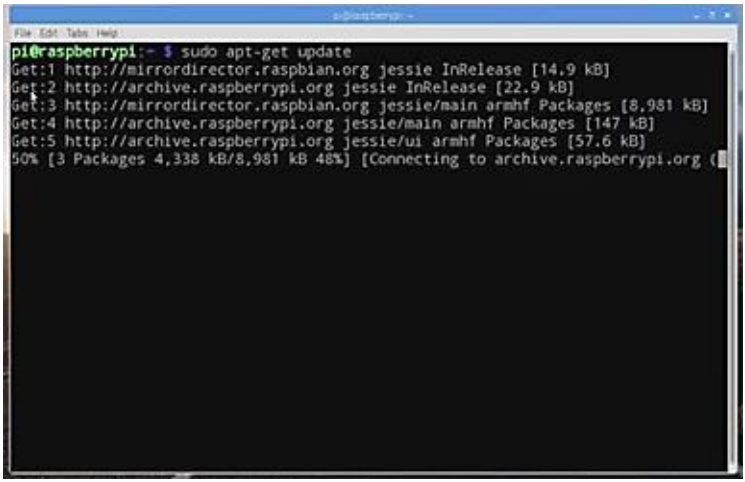
We already mentioned that the Raspberry Pi has several operating system packages and keeping these Operating Systems up to date is fundamental to the smooth operation of the device. A Wi-Fi network is required to download these updates. The Raspbian Operating System keeps a list of software packages that are available online for download. Before you download any software, it is a good idea to update this list in case the software package is outdated. To update these packages, we will be using a tool called APT,

which you can access from the terminal. To access the APT tool, we will use the "sudo" command which stands for "Super User Do" Command as shown below.

pi@raspberrypi:- $ sudo apt-get update

The 'sudo' command provides a way for users of Raspbian Pi operating system to temporarily increase their security privileges and perform software upgrades.

The above command line, when executed, connects the Raspberry Pi to various repositories and gathers information about software packages and updates. This process usually takes a while to complete, especially if you have not performed it in a while.



When it is done fetching the software index update, a statement comes up thus; "Reading package lists. . . Done"

After running the sudo command, you can run the "upgrade command" if you wish. What the "upgrade command" does is

to compare the version of the currently installed software to the ones in the software index update list. If there are any new versions in the list, it will automatically download and update them.

The "Upgrade command" is as follows

pi@raspberrypi:- $ sudo apt-get upgrade

You will be prompted to accept if you want to free or use the disk space on the memory card that this action would cause. Click on "Y" for Yes on the Keyboard and the upgrade process will commence. This will also take a while to complete. On the completion of the upgrade, you will have a fully updated Raspberry Pi.

Note: If you purchased an old NOOBS preloaded SD card, you should upgrade to the Raspbian version with Pixel. To do this, run the command;

pi@raspberrypi:- $ sudo apt-get dist-upgrade

Chapter Three

Programming Basics for Raspberry Pi Using Python

In this chapter, we will introduce the Python programming language. Python is a high-level object-oriented programming language, i.e., the codes written in Python is closer to the English Language; therefore, enhancing readability. It is one of the easiest programming languages in the market, and companies such as Google, Reddit, Dropbox, YouTube, Yahoo, and NASA widely use it as their main or support language. It was created by Guido Van Rossum in 1989 and was released in 1991. Along with the core functionality, Python supports using modules and packages, which mean we can import specificfunctionality when they are needed. Python has multiple versions – Python 1.0 introduced in January 1994, Python 2.0 launched in October 2000 and the latest, Python 3.0, which was introduced into the market in December 2008. The version 2.X is still actively in the market because when 3.X came into the picture, it was different from the version 2.X. When Java 8 came into the picture, it was backward compatible with Java 1.7, and when Java 1.7 came into the picture, it was backward compatible with version 1.6, but this is not the case with Python 3.0 and 2.0. The two versions are totally different; that is why we still have the version 2.X still relevantand available in the market. The version 2.X will have support until 2020 before they

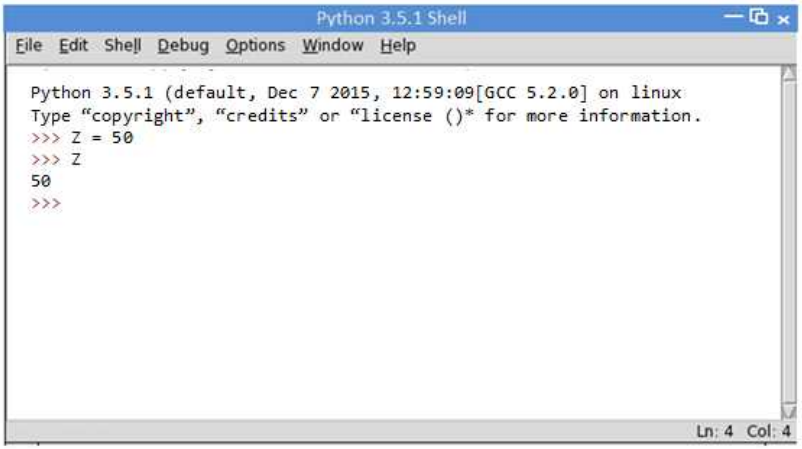
are finally fazed out. Python 3 programming language will used throughout this book for our projects. We can run Python on the Raspberry Pi by going to the Applications Menu – Programming – Python 3.

This brings up the python shell or IDLE where you can write python commands.

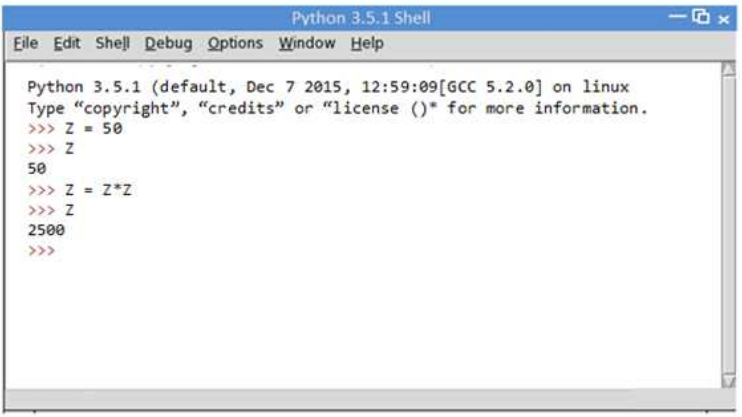
Let's look at some real test commands on python IDLE.

Variables

Variables act as a placeholder for whatever you place in them, and they can be changed. For instance, let us create a variable, say z equal 50 [>>>>z = 50], we can recall z by typing in z, and it will return 50.



You can also write a variable thus, z = z*z and if we call z in the next line, we have 2500.



Another way we can define the variable z is z = 55 + 32. When you call z, it returns 87. A variable can also contain a function. For instance, z = "whoa" and when it is called, it returns whoa.

Swapping Two Variables in python

Here we will look at how to swap two numbers in Python. Let's say we have two variables 'a' and 'b'. The value for 'a' is 5 and the value for 'b' is 6, and we want to swap these values. We have various ways of implementing this action in other programming languages and what is common in most of the languages is the use of a third variable to swap them. Same goes for Python programming Language; a third variable which is a temporal variable is introduced. For instance;

```
>>> a = 5
>>> b =6
We can't say;
a = b
b = a
```

If we do that, the output will be 6 because the moment you say a = b, this means the value for b which is 6, goes into 'a' and the original value for 'a' which is 5 will be lost. So what we do is, before assigning the value b to a, we will take the value of a and place it in a third variable let's call it 'temp,' then we will assign the value of 'temp' to b as illustrated below;

```
>>> a = 5
>>> b = 6
```

```
>>> temp = a
>>> b = temp
>>> print (a)
>>> print (b)
```

There is another way to swap values in two variables without the introduction of a third variable. Here we can use a formula. For instance,

```
>>> a = a + b        # answer is 11
>>> b = a – b        # value for a is 11 and new
                     value for b is 6. So new value for b is 5
>>> a = a – b        # new value for a is 6 and
                     value for b is 5 . So new value for a is
                     5
>>> Print (a)
>>> Print (b)
```

Another method we can use to swap variables without a third variable is the XOR function. Which is represented by ^

```
>>> a = a ^ b
>>> b = a ^ b
>>> a = a ^ b
>>> Print (a)
>>> Print (b)
```

The third option to swap two variables without introducing a third variable is given thus;

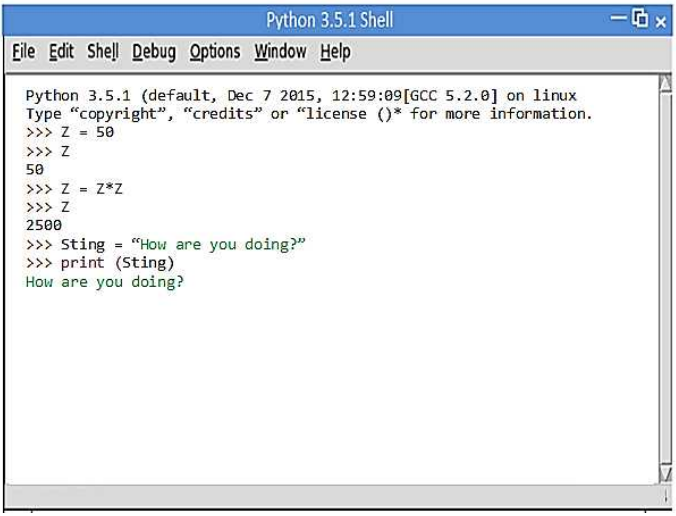
```
>>> a,b = b,a
>>> Print (a)
>>> Print (b)
```

What happens here is that the system will solve the right side a, b first, then b, a. The value for a is 4, and the value for b is 6, this is then moved into the stack and reverses using the concept of rotation called **ROT_TWO().** The value of 'a' comes to 'b,' and the value for 'b' comes to 'a.'

Strings

In programming, strings represent texts. However, in Python, there are several ways to create a string, as we have different types of texts. You can also create strings by declaring it as [string = How are you doing?] and when we invoke string, it will return the string. You can also create a string and store it in a variable, for instance [message = "Meet me this afternoon."] This stores the string 'meet me this afternoon' in a variable called 'message'. You do not have to type a semicolon at the end the way you do it in many other language like Java or C++.

```
>>> message = meet me this afternoon
>>> print (message)
>>> meet me this afternoon
```



Operators

In this section, we will look at how to use operators in a Python programming language. We will look at arithmetic operators, assignment operators, relational operators, logical operators, and unary operator.

Arithmetic Operator

Let's say;

```
>>> x = 2
>>> y = 3
```

We have two variables x and y and we will perform certain arithmetic operations with them as shown below;

```
>>> x = 2
>>> y = 3
>>> x + y
5
>>> x - y
```

```
>>> -1
>>> x*y
>>> 6
>>> x/y
>>> 0.6666666666666
>>>
```

Assignment Operators

Whenever you use the 'equal to symbol,' that is an assignment. Let's look at some examples of how assignment operators work.

```
>>> x = 2
>>> x = x + 2
4
```

We can do the above assignment operation using the increment function. Instead of saying x=x+2, we can say x +=2 which increments x by 2.

Example:

```
>>> x = 2
>>> x = x + 2
4
>>> x += 2
>>> x       #note that x was 4 initially and when
6           we increment by 2 we have 6
>>>
```

We can do same for multiplication,

Example

```
 >>> x = 6
>>> x *= 3
18
>>>
```

We can assign values in one line for two variables. For instance, we can assign two values (e.g., 5 and 6) to two variables (e.g. a and b) in one or two line(s). e.g.

```
>>> a = 5
>>> b = 6
Or we can assign it thus;
>>> a,b = 5,6
>>> a
5
>>> b
6
>>>
```

Unary Operator

Unary means one. If we have a value (e.g. 7) and we wish to negate it using unary operator, we say;

```
>>> n = 7
>>> n
7
>>> -n
>>> -7
>>> n = -n
>>> n
-7
```

Relational Operators

Relational operators allow us to compare different values. Here we look at less than <, greater than >, less than or equal ≤ to, greater than or equal to, etc. Let's illustrate a relational operators in Python using the two variables a and b.

```
>>> a = 5
>>> b = 6
```

Check if a is less than b

```
>>> a = 5
>>> b = 6
>>> a < b
True
>>> a > b
False
>>>
```

We can also check if two variables are the same. Note that we can not use the "equal to symbol" to do this because it is a symbol for assigning values to a variable. To compare two variables, we will use the 'double equal to (==)' symbol.

Example:

```
>>> a = 5
>>> b = 6
>>> a == b
False
>>>
```

```
>>> a = 7
>>> b = 7
>>> a == b
True
>>>
```

In the next examples, we will be checking if 'a' is less than or equal to 'b' and if 'a' is greater than or equal to 'b.'

```
>>> a = 7
>>> b = 7
>>> b <= b
True
>>>
```

```
>>> a = 7
>>> b = 7
```

```
>>> b => b
True
>>>
```

We can also check if the two variables are not equal. Here we use the not equal operator '!='. Let's say

```
>>> a = 8
>>> b = 7
>>> a != b
True
>>>

>>> a = 7
>>> b = 7
>>> a != b
False
>>>
```

Logical Operators

To understand logical operators, you need to understand the concept of true and false. You use logical operators when you have two conditions, and you intend to combine these conditions. It could be based on 'And,' 'Or' or 'Not' operators. Example: If we have two variables where a is 5 and b is 4; if we want to ensure that a is less than 8 and b should be less than 5; In this case, we will write two conditions, first, 'a' is less than 8 'and' b is less than 5. Note that the 'and' logical operator is placed in-between the two conditions. We can write the code thus;

```
>>> a = 5
```

```
>>> b = 4
>>> a < 8 and b < 5
True
>>>
```

Realized that both conditions are 'True' hence the operation is returned 'True'. We can write a code to check if a is less than 8 and b is less than 2.

```
>>> a = 5
>>> b = 4
>>> a < 8 and b < 5
True
>>> a < 8 and b < 2
```

Note the first condition 'a < 8' is true, and the second condition 'b < 2' is false. As long as one of the conditions is False, then the operation will return false.

```
>>> a = 5
>>> b = 4
>>> a < 8 and b < 5
True
>>> a < 8 and b < 2
False
>>>
```

To understand logical operations, the truth table is required. We have different truth tables for 'And', 'Or' and Not' We will represent True with one (1) and False with Zero (0) on the truth table while x and y represent two different conditional outputs. Below is the 'AND' truth table.

x	y	c
0	0	0
0	1	0
1	0	0
1	1	1

For 'AND' truth table, both conditions need to be true for the output to be true.

Let's take the 'OR' truth table. For this table, we need at least one condition to be true, and the output will be returned 'True.'

x	y	c
0	0	0
0	1	1
1	0	1
1	1	1

```
>>> a = 5
>>> b = 4
>>> a < 8 or b < 2
True
>>>
```

The 'Not' logical operation basically reverses your output. Example '

```
>>> x = True
>>> x
True
>>> not x
False
>>> x = not x
```

```
>>> x
False
>>>
```

Binary Format

Here, we look at the binary format and other number systems in python. Normally, when you work in programming, we do that in binary format and decimal system. Apart from these, we have two more systems – octal and hexadecimal. For instance, in networking, we talk about MAC and IP address, which are defined in hexadecimal format. What we will do in this section is converting these formats from one system to another with the help of python codes. You can convert a number from decimal system (base 10) to the binary system (base 2) using a function called bin. Note that in a binary system you can only go from zero to one (0 – 1) while in a decimal system, we can go from zero to nine (0 – 9). Octal system (base 8) starts from zero and ends at seven (0 – 7). On the other hand, we have the hexadecimal system (base 16) which goes from zero to nine (0 – 9), and after nine it takes the form of A – F. Example: Convert 25 into a binary number.

```
Python 3.7.0 (v3.7.0:1bf9cc5093, June 27 2019, 04:06:47) [MSC v.1914 32 bit (intel)] on win32
Type "copyright","credits" or "license()" for more information.
>>> bin(25)  #click on 'Enter Key" on the Keyboard
'0b11001'    # '0b' indicates binary format.
>>>
```

Example 2: Convert 0b0101 to decimal

```
>>> 0b0101
5
>>>
```

Example 3: Convert 25 to octal

```
>>> oct(25)
‘0o31’      # ‘0o’ indicates octal format
>>>
```

Example 4: Convert 25 to hexadecimal

```
>>> hex(25)
‘0x19’      # ‘0x’ indicates hexadecimal format
>>>
```

Example 5: Convert 10 to hexadecimal

```
>>> hex(10)
‘0xa’       # ‘0x’ indicates hexadecimal format
>>>
```

Example 6: Convert '0xf' to decimal

```
>>> 0xf
15
>>>
```

Bitwise Operators

Binary operations are used extensively in Bitwise operations

Bitwise operations are divided into six operators –

Complement (~) operator

Bitwise And **(&)**

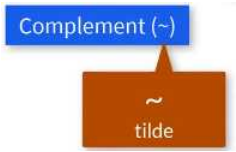
Bitwise Or **(|)**

XOR **(^)**

Left Shift (<<) and
Right Shift (>>)

Bitwise Complement (~) Operator

Another name for the complement operator is the tilde operator due to its symbol. The tilde symbol is just below the "ESC" key on the keyboard.



What a compliment operator does is the reverse of binary format. For instance complement one equal zero (~ 1 = 0) and complement zero equal 1 (~ 0 = 1). Complement 12 is – 13 (~12 = -13). When we convert 12 to binary, we have 00001100. A Complement of the binary of 12 (i.e., reverse the numbers in the binary of 12) is 11110011 which is -13. This means wherever you have zero, replace it with a one and wherever you find 1, replace it with a zero. Now, we have a concept of two's complement (2') in our system, which allows us to store only positive numbers. The question is, how do we store negative numbers? To store negative numbers, we must first convert it to positive number, and we can only do that with the help of two's complement (2'). To find two's complement, we first of all, find one's complement plus one (1' + 1). Let's look at the steps to find one's complement.

Step1: Change the negative number to positive and convert it into binary format

So, -13 will be 13 and the binary format of 13 = 00001101

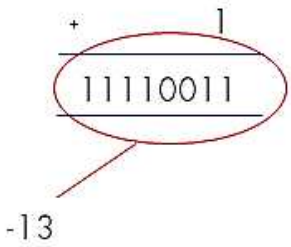
Step 2: Find two's complement of the binary format

2' of 00001101.

Note that 2' = 1' + 1

1' of 00001101 = 11110010.

2' of 00001101 = 11110010



11110011, when compared to the complement of 12, are the same (11110011); therefore complement of 12 is (- 13).

Example:

```
>>> ~ 12
-13
>>>
```

Bitwise AND (&) Operator

In logical operators, we talked about 'AND' and 'OR.' Here we will look at Bitwise AND. In Bitwise AND like logical AND operator, an output is true only when both conditions are true.

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

We then use 'AND' for bitwise and because it is reserved for logical 'AND.' For bitwise AND we use the ampersand symbol (&).

Example: What is the bitwise operation between 12 &13? What this operation does is to convert 12 and 13 into binary format and compare each binary digit. Note that if both are 1 the output will be 1, but if one of the digits is 1 and the other is zero, the output will be zero following the Logical and Bitwise AND rule. So we have;

12 - 00001100

13 - & 00001101

00001100 ⟶ 12

The output of 12 & 13 is 12

Using Python code we have;

```
>>> 12 & 13
12
>>> 25 & 30
24
>>>
```

Bitwise OR (|) Operator

In Bitwise 'OR' like a logical OR operator, an output is true only if one conditions is true.

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

We use | for bitwise OR .

Example: What is the bitwise operation between 12 or13? What this operation does is to convert 12 and 13 into binary format and compare each binary digit. If we have 1 (true), then the output will be 1(true) irrespective of the other binary digit is 1 or 0 (true or false) following the Logical and Bitwise 'OR' rule. So we have;

```
12    -        00001100
13    -      | 00001101
                00001101  ⟶  13
```

```
>>> 12|13
13
>>>
```

XOR (^) Bitwise Operator

A cap symbol (^) represents the XOR operator. In XOR bit-wise operation, the rule says if we have an odd number of 1, the output is one, and when we have an even number of 1, the output is zero. In other words, if both numbers are different, the output is one (1), but if both numbers are the same, the output is zero.

XOR

x	y	xy
0	0	0
0	1	1
1	0	1
1	1	0

Example: What is the bitwise operation between 12 XOR 13.

```
12  -    00001100
13  -  ^ 00001101
         --------
         00000001  ⟶  1
```

```
>>> 12^13
1
>>> 25^30
7
>>>
```

Left Shift (<<) Operator

In left shift as the name implies, we shift the binary digit to the left-hand side. For instance, if we left-shift 10 (1010) by 2, it means we are adding two zeros to the right-hand side. Note that the binary digit of 10 is 1010 if we perform a left shift operation of 10 by 2 bits (10<<2); we add two zeros to the right therebypushing 1010 to the left. A left-shift in 1010 (10) will result in 101000 (40). In left shift, you gain bits.
Example:

```
>>> 10<<2
40
>>>
```

Right Shift (>>) Operator

In right shift, we lose bits. We will lose the last two binary digits to the right-hand side. Let's consider the right shift operation 10>>2. This means we will lose the (10) to the left-hand side of the binary digits of 10 (1010). When we do this, we will be left with 10, which is 2

```
>>> 10>>2
2
>>>
```

Print

The print command is used to output information on the screen. For example, >>>> print {'How are you doing?'} and it will return the word 'How are you doing?' without the quotation mark. One way to differentiate between a string and a text printed to screen is that strings are always in quotation marks while a text printed to screen does not come in quotation marks.

Python Data Types

When you work on a project, it is imperative to understand what type of data you are working with, so you can process the data in a particular format to prevent bugs. There are several data types in Python programming language, namely - None, Numeric, List, Tuple, Set, String, Range and Map or Dictionary.

None

If you have a variable which is not assigned to any value, it is referred to as None. In other languages, we use the 'null' keyword, but in Python, we use none.

Numeric Type

We have several numeric types, for instance – int, float, complex and bool. Where int stands for integer, float for decimal numbers, complex for complex numbers and bool stands for Boolean.

Float

We can say `>>> num = 2.5, where 2.5 is a float. How do we know the number is a float?`

```
Let's say;
>>> num = 2.5
>>> type (num)
<class 'float'>
>>>
```

When we use 'type' as a function and parse num, it will return the class as float.

If we say;

```
>>> num = 5
>>> type  (num)
<class  'int'>
>>>
```

For the above, class changes to 'int' because the variable is an integer.

Complex

In mathematics, we have the concept of complex numbers

$$a + bi$$

if we say;

```
>>> num = 6+9j
```

```
>>> type(num)
<class 'complex'>
>>>
```
Here class is returned as a complex number.

Let's say we want to convert a float value to integer.
```
>>> a = 5.6
>>> b = int (a)
```

Here we take a variable called 'a' and assign 5.6 to the variable 'a'. We have another variable 'b.' If we want to assign 'a' as an integer, we will be using the 'int' function in which we will parse 'a' b = int (a). This 'int' function will convert the float value or any value into integer format as shown below.

```
>>> a = 5.6
>>> b = int (a)
>>> type  (b)
<class  'int'>
>>> b
5
>>>
```

We can also convert an 'int' value to float

```
>>> k = float(b)
>>> k
5.0
>>>
```

We can convert a normal number to complex numbers
```
>>> k = 6
>>> c = complex  (b,k)
>>> c
(5+6j)
>>>
```

It says 5+6j where 5 is your b and 6 is your k.

Boolean

When it comes to Boolean, it means true or false. In computing, we make decisions based on true or false, depending on the conditions applied. For instance, if we want to check from the above functions if b is less than k. When we hit the enter button it returns the Boolean type true or false.

```
>>> b<k
True
>>>
```

It returned 'True' because b is less than k. Remember that b is 5 and k is 6. So 'True' is a Boolean type. You can also assign Boolean to variables as illustrated below

```
>>> z = b<k
>>> z
True
>>>
If we want to determine the variable class type thus;
>>> z = b<k
>>> z
True
>>> type (z)
<class 'bool'>
>>>
Let's look at a Boolean function that returns 'False'.
>>> z = b>k
False
>>>
```

Another thing you should note is that in python, True is represented by 1, and False is represented by 0. Let's look at the function below;

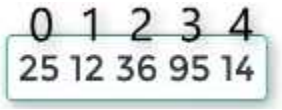
```
>>> int (True)
1
>>> int (False)
0
>>>
```

List

If you have different values (e.g., int, strings, float, etc.) and you wish to group them, you use 'List.' A list in Python can hold the same or different data. Here is how you work with a list data type. Firstly, we will find out the class data type. Let's say we have some numbers 25, 36, 45, 12, and we wish to identify if it is a list data type or not. It returns a class of type 'list' if it is a list. Note that we enclose the values in the list in angular brackets.

```
>>> lst = [25,12,36,95,14]
>>> type(lst)
<class  'list'>
```

Values in a list are also index, and we can use these index numbers to call items in a list. For instance, if we have a list of numbers, 25,12,36,95,14, the first element is indexed in position 0, the second element is indexed in position 1, the third element is indexed in position 2, the fourth element is indexed in position 3, etc. as shown below;



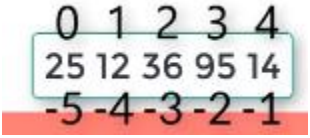
So, we can call the elements in different positions in the list by their index number. For instance; if we use the function nums = [0], it returns 25, which is the element in index number 0.

```
>>> nums = [0]
25
>>> nums = [4]
14
>>> nums = [2]
36
>>>
```

If we wish to print elements starting from a particular index number, let's say we want to print the element occupying index number 2 to the end, we make use of colon as shown below;

```
>>> nums[2:]
[36,95,14]
>>>
```

Index numbers for elements in a list are also numbered in negative range when we are taking them from the end backward as illustrated below;



So, if we say num [-1], the output will be 14.

```
>>> nums[-1]
14
>>> nums[-5]
95
>>>
```

We can also have lists with strings alone e.g.

```
>>> names =  ['martins', 'smith', 'John']
We can also have a list with different data, for
instance, float, string and integer.
>>> values = [9.5, 'smith', 24]
```

Working With two Lists

We can have two separate lists working together for instance,

```
>>> nums =  [25,12,36,95,14]
>>> names =  ['martins', 'smith', 'John']
>>> twolists =  [nums, names]
>>> twolists
[[25,12,36,95,14],['martins', 'smith', 'John']]
```

Note that the first list is 'nums' and the second list is 'names.' We brought both lists together and named them 'twolists.' When we called the variable 'twolists', it returned `[[25,12, 36,95,14], ['martins,' 'smith,' 'John']]` as output, one of numbers and the other of string, thereby bringing both lists together.

Another amazing thing about lists is that we can perform certain operations (such as append, clear, copy, count, extend, index, insert, pop, remove, reverse, del, sort, etc.) on

them. They are also 'mutable'; this means we can change their values.

Append

For instance, lets append 45 to our existing list [25,12, 36,95,14], we use the function;

```
>>> nums =  [25,12,36,95,14]
>>> nums.append(45)
>>> nums
[25,12,36,95,14,45]
>>> nums
```

Insert

We can also use 'insert' to add element(s) to an existing list. However, there is a difference between append and insert. When we use append, the new element(s) is appended at the end of the list while for 'insert,' the element(s) is placed at any index value position we wish. This means when inserting an element, we must specify the position of the index value we wish to place it as well. For instance, Let's place 77 at the position of index value 2. We use the function;

```
>>> nums =  [25,12,36,95,14,45]
>>> nums.insert(2,77)
>>> nums
[25,12,77,36,95,14,45]
```

When we print num, you will observe that 77 has taken the position of index number 2.

Remove

The remove operation, as the name implies, removes number(s) from the list. For instance, if we wish to remove 14 from our list, we can use the function below;

```
>>> nums = [25,12,36,95,14,45]
>>> nums.remove(14)
>>> num
[25,12,77,36,95,45]
```

Pop

We can delete a value from a list based on their index number using the 'pop' operation. For instance, let's delete the 12 from the list using its index value. Note that 12 is under index number 1. We use the function;

```
>>> nums = [25,12,36,95,14,45]
>>> nums.pop(1)
12
>>> nums
[25,77,36,95,45]
```

We can also use the 'pop' operation without an index value. You can use this when implementing a stack in data structures. Stack in data structure talks about 'Push' and 'Pop.' Push means pushing the element in the stack and pop means removing the element from the stack. When we say pop, it removes the last element added, which is also referred to as 'Last In, First Out' (LIFO). We also have the FIFO in a stack data structure which means 'First in, First Out.' What happens when we do this is that it removes the last element added to

the list and in this case, the last element added to our list is 45. So when we pop without an index value we have;

```
>>> nums =  [25,12,36,95,14,45]
>>> nums.pop()
45
>>> nums
[25,12,77,36,95,14]
```

Del

We use the 'Del' operation when we want to remove multiple values from a list. If you want to use the 'del' operation, you must specify the index values of the elements you want to remove from the list. You can delete elements starting from index value 2 to the end.

```
>>> nums =  [25,12,36,95,14,45]
>>> del nums[2:]
>>> nums
[25,12,]
>>>
```

We will leave only two values in the list, the 0^{th} and 1^{st} and delete everything from the 2^{nd} to the end.

Extend

The 'Extend' operation is used to add multiple values to a list. To do this, type nums.extend and in the bracket, specify multiple values that should be added to the list. For instance, we want to add 29, 12, 14 and 36 to the existing list, we write it thus;

```
>>> nums =  [25,12,36,95,14,45]
```

```
>>> nums.extend([29,12,14,36])
>>> nums
[25,12,36,95,14,45,29,12,14,36]
>>>
```

Note that the values to be added to an existing list must be in square brackets.

Sort

We use the sort function to arrange the elements in the list in ascending order. For instance,

```
>>> nums =  [25,12,36,95,14,45]
>>> nums.sort()
>>> nums
[12,14,25,36,45,95]
>>>
```

Clear

This clears the entire elements in the list.

```
>>> nums.clear
```

Inbuilt Functions

In Python, we have several inbuilt functions such as min, max, sum, etc.

Min

This Min is used to find the minimum values in a list. Let take for example; we want to find the minimum values in our list;

```
>>> nums =  [25,12,36,95,14,45]
>>> min  (nums)
12
```

```
>>>
```
The minimum value from our list is 12.

Max

The max is used to find the maximum value in a list. Let's find the maximum value in our list.

```
>>> nums =  [25,12,36,95,14,45]
>>> max  (nums)
95
>>>
```

Sum

We can find the sum of all the values in the list using the sum function. For instance;

```
>>> nums =  [25,12,36,95,14,45]
>>> sum (nums)
227
>>>
```

Set

Set is a collection of unique elements. We can perform certain operations carried in 'List' with 'set'. For instance, Set supports add, copy, clear, difference_update, discard, intersection, intersection_update, isdisjoint, issubset, pop, symmetric_difference, symmetric_difference_update, remove, union, difference etc. However, indexing is not supported in set. It is important you use a curly bracket to hold set values. For instance,

```
>>> s = {25,36,45,15,12,25}
>>> s
```

```
{36,12,45,15,25}
>>>
```

When a set is called, it does not repeat values that appear twice. e.g., 25 is returned once. The class type is returned as 'set'

```
>>> s = {25,36,45,15,12,25}
>>> s
{36,12,45,15,25}
>>> type(s)
<class 'set'>
>>>
```

Tuple

The tuple is similar to the list because both have a collection of values, but the difference between a list and tuple is that in a list, we can change specified values because it is mutable. However, Tuple is immutable, that is we cannot change its values because it doesn't support item assignment. Another difference between a tuple and a list is that iteration in a tuple is faster than in a list. The tuple is used in projects where a given number of elements are not expected to change, and you require an enhanced speed of execution. Finally, unlike the list whose values are enclosed in square brackets, values in tuple data type are enclosed in a normal bracket. For instance,

```
>>> t = (25,36,4,57,12)
>>> type(t)
<class 'tuple'>
>>>
```

We can only use two operations on tuple – Count, and Index. The Count operation, as the name implies, will count the number of occurrence of a number and the index operation is used to identify the index number of elements in the tuple.

Array

Arrays are similar to a list, but the only difference is that an array holds values of the same type. If an array is an integer array, then all elements must be integers. Float array only holds float values. Arrays in Python don't have a specific or fixed size, which means they can be increased or reduced. For instance, if you have an array with five values, you can decide to increase it to ten values, or if you have an array with fifteen values and you wish to reduce it to four values, you can do that as well. This makes it flexible to work with arrays. It also offers programmers the opportunity to work with several operational methods, e.g., find, index, pop, etc. You may ask, when can we use arrays – let's say you have a list of ten students in a class and each of them has a score in an examination on a subject. You will need to create ten variables, for instance, mark 1, mark 2, mark 3 up to mark 10. This becomes difficult if you have 100 students or above—you can't create 100 variables. Instead of doing that, we can create an array. To use an array, we need to import a module known as 'array.' To do this, we say;

```
Import array as arr
Arr.array[]
```

You can specify the functions you want to work with or simply specify asterisks *. Specifying asterisks means you want to work with all the functions.

```
from array import *
```

Note that when writing an array, we must specify two things - the type and the values. Why type? This is because all values in an array must be of same type (int, float, etc) and every type has a code as shown below;

TypeCode	C Type	Python Type	Min. size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

For integers, we have different types - byte, long, and int. The moment you change the type, it changes the size in the memory. For example, a small integer takes only 1 byte; a normal integer takes 2 bytes while a long integer takes 4 bytes. Float values takes 4 bytes; double float gives more memory of 8 bytes. If we want to go for a character, we use 'U' which stands for Unicode, and it takes 2 bytes. Now, we

look at signed and unsigned integer. A normal integer goes from a negative range to a positive range, but if you don't want to store or consider a negative value, you will go for an unsigned integer because it starts from zero and ends in a positive value. If you want to work with an unsigned integer, you have to use a capital 'I'. Use small 'i' for a long unsigned integer.

Example

Let's create an array of integers called 'vals' which means values.

```
from array import *
vals = array ['i',[5,9,8,4,2]]
```

We can print the above arrays using print(vals). When we run it, it will output arrays of type integer and the values.

```
from array import *
vals = array ['i',[5,9,8,4,2]]
print(vals)
array  ['i',[5,9,8,4,2]]
```

If we add a float say 8.5 in the array declared as type int, if we try to print, it will return a syntax error message.

```
from array import *
vals = array ['i',[5,9,8.5,4,2]]
print(vals)
TypeError: integer argument expected, got float
```

However, we can add negative values in an array declared 'i' because 'i' means signed integer which accommodates negative and positive values.

```
from array import *
vals = array ['i',[5,-9,8,4,-2]]
print(vals)
array  ['i',[5,-9,8,4,-2]]
```

But if we declare the array of type capital 'I', adding a negative value to the array will return an error message. For example;

```
from array import *
vals = array ['I',[5,-9,8,4,-2]]
print(vals)
overflowError: can't convert negative values to
unsigned int.
```

In summary, when you create an array, be sure to specify a proper type code.

Arrays also have several functions such as

buffer_info(self, args, kwargs) ArrayType
byteswap (self, args, kwargs) ArrayTpe,
frombytes (self, args, kwarg.. ArrayType,
tobytes (self, args, kwargs) ArrayType,
getattributes(self, args, ArrayType
_init_subclass_(clsa) object ArrayType
append (self, args, kwargs) ArrayType
remove (self, args, kwargs) ArrayType
reverse (self, args, kwargs) ArrayType
index (self, args, kwargs) ArrayType
typecode ArrayType

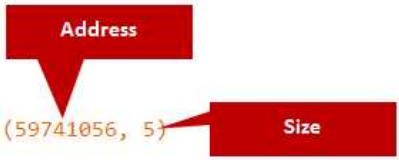
itemsize ArrayType
count (self, args, kwargs) ArrayType
extend (self, args, kwargs.. ArrayType etc

buffer_info

For instance, buffer_info will give us the size of the array. For Example;

```
from array import *
vals = array ['i',[5,9,-8,4,2]]
print(vals.buffer_info())
(59741056, 5)
```

This outputs a tuple (59741056, 5) where the first parameter is the address of the array and the second parameter, is the size of the array.



If we change the number of elements in the array to 4, for instance, `vals = array ['i',[5,9,-8,4]], buffer_info function will give us (16303920, 4),` where the first parameter is the address of the array and the second parameter is the size of the array.

typecode

The 'typecode' function prints the type of code with which you are working. For instance, if we `print(vals.typecode())` `for the array [5,9,-8,4]],` it will return small 'i', indicating integer.

```
from array import *
vals = array ['i',[5,9,-8,4,2]]
print(vals.typecode())
i
```

append

We use the append function to add values to an array.

remove

The 'remove' function as the name implies, remove elements from an array.

reverse

If you want to reverse the entire array, you use the reverse function. For instance;

```
from array import *
vals = array ['i',[5,9,-8,4,2]]
```

```
vals.reverse()
print(vals)
array  ['i',[2,4,-8,9,5]]
```

index

We can use the index to print the elements in an array one after the other. For instance, we can print the element in index value 0, which is 2. We use the function thus;

```
from array import *
vals = array ['i',[2,4,-8,9,5]]
print(vals[0])
2
```

```
from array import *
vals = array  ['i',[2,4,-8,9,5]]
print(vals[1])
4
```

We can also print the entire values one after the other with the aid of a loop. We can either use a 'while' or 'for' loop. Note that the print is indented for the loop to work properly. The 5 in the range indicate the number of elements in the array. That is, the loop will start at index number 0 and end at index number 4.

```
for i in range  (5):
    print(vals[i])
5
9
-8
4
2
```

If we don't know the length/number of elements in an array, we can make it more dynamic by writing the loop thus;

```
for i in range  (len(vals):
      print(vals[i])
5
9
-8
4
2
```

Where len = length and len(vals), means length of values. Length of vals gives us the length of the array, and we can parse this length in range.

We can also create new array with the same values. For instance we have an array [5,9,8,4,2] and we want to create a new array (newArr) using the existing array; we say

```
Vals = array ('i',  [5,9,8,4,2])
newArr = array(vals.typecode, [a for a in vals])
      for a in newArr:
            print (a)
5
9
8
4
2
```

What this line `(vals.typecode, [a for a in vals])` indicate is,

(1) `vals.typecode` - get the type from the old array which is vals.

(2) `[a for a in vals]` – this is actually a loop it says "take one value at a time from vals". This will fetch the values one after the other and assign it to the new array. You can als assign the square of each value in vals into the new array. In this case, we can say [a*a for a in vals]) and it will fetch the square of each value in the old array into the new array when we run the code. Let's write the function;

```
Vals = array ('i',  [5,9,8,4,2])
newArr = array(vals.typecode, [a*a for a in vals])
      for a in newArr:
            print  (a)
25
81
64
16
4
```

We have seen how to work with integers in an array; we will now look at using characters in arrays. In arrays as a character, we declare the array type as 'u' which stands for Unicode character

```
from array import *
vals = array  ['u',['a','e','i']]
for i in range (len(vals):
      print(vals[i])
a
e
i
```

In our previous examples, we have seen how to print elements of an array using the 'for' loop. Let's look at printing the elements of an array using the 'while' loop. The difference between using the 'for' loop and 'while' loop in printing arrays, is that for the 'for' loop, we make use of two steps, while for that of 'while' loop, we make use of three steps – the initialization, check for condition and increment. For instance

```
i = 0
while i<len(newArr):
      print(newArr[i])
i+=1
25
81
64
16
4
```

However, it is much easier to use the 'for' loop because no initialization, increment/decrement or checking for the conditions is required.

Arrays with Values from the User

In this section, we will look at an array where the values will come from the user, that is, an array whose size is not known. In this case, we will create a blank array, and the user will be asked to enter the values. Here are the steps.

Step 1: we create a blank or empty array called 'arr'

```
from array import *
arr = array ('i', [])
```

Step 2: First we add a line that would ask the user the length of the array (i.e., the number of elements in the array),

```
n = int(input("Enter the length of the array"))
```

Once the user enters the length of the array, the next step is to ask the user to enter value they want to insert in the array. This has to be done repeatedly. This keeps prompting users to enter values one after the other. Note that when we want to ask for something to be done repeatedly in a Python programming language, a 'for loop' is ideal even though a 'while loop' can still do this. We will use a 'for loop' and a variable 'i' which is a range (i.e., 0,1,2,3,4....n). Every time this loop runs, and it ask a user to enter a value, it will insert the value in a variable 'x,' and the values in x is then sent into the array using the append function as illustrated below;

```
for i in range(n):
    x = int(input("Enter the next value"))
    arr.append(x)
```

Step 3: Let's bring the entire code together and run it.

```
from array import *
arr = array ('i', [])
n = int(input("Enter the length of the array"))
for i in range(n):
    x = int(input("Enter the next value"))
    arr.append(x)
print(arr)
```

When this code is run, let's say we want an array of 6 elements 14, 15, 17, 20, 39, 40. This is what we get;

```
Enter the length of the array 4
Enter the next value 14
Enter the next value 15
Enter the next value 17
Enter the next value 20
Enter the next value 39
Enter the next value 40
Array  ('i',  (14, 15, 17, 20, 39, 40))

Process finished with exit code 0
```

We can also write a code for the user to enter a value, and we will search and print for the index number. There are two ways to print the index number of an array inserted by a user – manually and using some inbuilt functions. Let's say a user after inputting the values 14, 15, 17, 20, 39, and 40 manually into an array, and he intends to know the index number of 20 in the array. What we do is to write a function that has to compare elements in the array and the individual index number. For instance, with 20, what the function does is ask "is 20 the same as the first value of the array? If it is the same, we get the index number else it shifts to the next value to see if it matches. If the answer is no, then shift to the next value. You do this until it gets to 20 which is the fourth element in the array with an index number of 3. Every time we run the loop, we have to increment the value by one until you get to the index number for the element in the array you are searching. Let's look at the code line for this function;

```
val = int(input  ("Enter the value for search"))
```

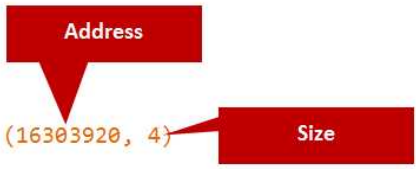
```
k = 0
for e in arr:
      if i==val:
          print(k)
          break
      k+=1
```

When we run this code we have

```
Enter the length of array 6
Enter the next value 14
Enter the next value 15
Enter the next value 17
Enter the next value 20
Enter the next value 39
Enter the next value 40
array ('i', (14, 15, 17, 20, 39, 40)
Enter the value for search 17
2
Process finished with exit code 0
```

17 has index number 2. If we enter 39, it will return 4, which means 39 has index number 4. Note that the "e" in the 'for loop' is not coming from a range but from the array itself. In line 2 of the code, k is a counter variable which represents the index number in this case. Every time the loop iterates, we will increment the value of k by one. If the element is not matching, we keep incrementing the value of k, until k matches with the element in question then we introduce the 'break' function which ends the iteration and we print k. If you intend to use another function in place of the 'for loop' simply type the code.

```
print(arr.index(val))
```



String

String data type (str) is enclosed in double or single quotes for instance,

```
>>> a = "smith"
>>> type  (a)
<class  'str'>
>>>
```

Range

The Range function can only be found in Python programmming language. It is useful when iterating between values. For instance, if you are moving from 1 - 10 or 10 - 100.
Example: if we want a range of 0 - 10, we define it as

```
>>> range (10)
Range (0, 10)
>>> type(range(10))
<class 'range'>
>>>
```

We can output a range by converting it to a 'list" as illustrated below;

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

We can also have a range with different iterations. For example, if we want to print all even numbers from 1 – 10, we say; range (2,10,2) – this means list a number 1 to 10, starting at 2 and end with 10 using intervals of 2. This will list all the even numbers between 1 and 10 as shown below;

```
>>> list(range(2,10,2))
[2, 3, 4, 6, 8]
```

Mapping or Dictionary

Dictionary is used to fetch data efficiently or faster from a huge amount of data. With every value, you assign a key. Normally, we assign an index number for each value, but in a dictionary, for every value, we assign a key. For instance, let's say we want to have all the mobile phone names. Let's say in my friend's group, I have fifty phones and everyone has a different phone, we can assign an index number to each phone. iPhone x could take index number 0, Index number 1 for Samsung S10, Index Number 3 for iPhone XR, etc. Now instead of using an index number, we could use a key, for example, each phone is assigned the name of the owner. iPhone X has the key Michael, Samsung S10 has the key Kelvin, iPhone XR has the key Victoria, etc. In this case, every mobile phone has a name (key) attached to it, which makes it easier to fetch. Keys can also be whole numbers instead of strings. For instance, students in a university identified by their matriculation number, that becomes the key. One important thing to note is that the keys

should be unique. If you have a dictionary, the entire key must be uniquely assigned. Also, note that dictionaries are defined by curly brackets; this is because like 'set,' keys are unique. For instance, if we have a dictionary of names of people and the type of mobile phone they use. We can define this dictionary thus,

```
>>> d = {'Smith':'Samsung S10','Victoria': 'Iphone X', 'Michael' :
        'Iphone XR'}
>>> d
{'Smith':'Samsung S10','Victoria':'Iphone X', 'Michael':'Iphone XR'}
```

In this case, Smith, Victoria and Michael are keys. This is because when you print the keys using the function `d.keys ()`, it will output Smith, Vitoria and Michael

```
>>> d = {'Smith':'Samsung S10','Victoria':
'Iphone X','Michael' : 'Iphone XR'}
>>> d
{'Smith': 'Samsung S10','Victoria':'Iphone X',
 'Michael':'Iphone XR'}
 >>> d.keys ()
dict keys {['Smith', 'Victoria', 'Michael']}
>>>
```

```
We can get the values of the dictionary using
d.values()
```

```
>>> d = {'Smith':'Samsung S10','Victoria': 'Iphone
X','Michael' : 'Iphone XR'}
>>> d
{'Smith': 'Samsung S10','Victoria':'Iphone X',
'Michael':'Iphone XR'}
 >>> d.keys  ()
dict keys {['Smith', 'Victoria', 'Michael']}
>>> d.values()
```

```
dict values {[‘Samsung S10’ , ‘Iphone X’ , ‘Iphone XR’]}
>>>
```

If you want to get a particular value, you would need to use a key. For instance, if we want to know what phone Smith is using. We say;

```
>>> d.[‘smith’]
‘Samsung’
>>>
```

Note that we used a square bracket here to fetch the content of a key. Another way to use a key without using the square bracket is with the use of the 'get function'. For instance;

```
>>> d.get{‘smith’}
‘Samsung’
>>>
```

Import Math Functions

In Python we have a lot of modules to work with which are not available to you by default. Before you can utilize these modules, you need to ask for them using the import function. For instance, when carrying out calculations, we must import mathematical functions before we can use them. We can also use mathematical constants like pi, e etc Here is what happens when we perform a mathematics calculation without importing the math function;

```
 >>> x = sqrt(81)
Trackback (most recent call last):
       File”<psyhell#0>”,line 1,in<module>
```

```
        X = sqrt(81)
NameError:name  'sqrt' is not define.
```

It returns an error message. To address this problem, we import module math. Math is a module that has all the mathematics functions.

```
>>> import math
>>> x = math.sqrt(81)
>>> x
9.0
>>>
```

Floor and Ceil Function

Floor and ceil functions are used in rounding off decimal numbers. The floor will give you the least integer while ceil function will give you the highest integer. For example, if we have a decimal number 2.5, we can use the floor function to round the value off to 2 irrespective of whether the values are between 2.0 to 2.9 while we use the ceil function to round off values between 2.0 to 2.9 to 3.

```
>>> import math
>>> x = print(math.floor(2.9))
>>> 2
>>>

>>> import math
>>> x = print(math.ceil(2.9))
>>> 3
>>>
```

Power Function

We can perform power operation in two ways. Firstly let's say we want to find 3 to the power of 2, we can say;

```
>>> 3**3
9
>>>
```

Another way to carry out power operation is with the use of the power function

```
>>> import math
>>> x = print(math.pow(3,2))
>>> 9.0
>>>
```

Mathematical Constants

```
>>> import math
>>> print(math.pi)
3.141592653589793
>>>
```

```
>>> import math
>>> print(math.e)
2.718281828459045
>>>
```

Concept of Alias

Instead of constantly using 'math' in our code, we could introduce the concept of an alias to abbreviate it. For instance, we can replace maths with 'm' or any other variable. Let's illustrate;

```
>>>import math as m
```

The above code line simply means "I am importing math, but within the code, I will be using 'm' in place of math.

```
>>>import math as m
>>>m.sqrt(81)
9.0
>>>
```

Importing Specific Math function

Whenever we import math, we import the entire mathematical functions and classes. If you don't intend to import the entire math module but want a specific math module. For example you only need the power function or factorial etc. In this case, what you will do is;

```
>>>from math import sqrt, pow
```

In this scenario, you don't have to specify math because you have imported the function you need.

```
>>>from math import sqrt, pow
>>>pow(4,5)
1024.0
>>>
```

You can get more mathematical functions you can try out with 'help'.

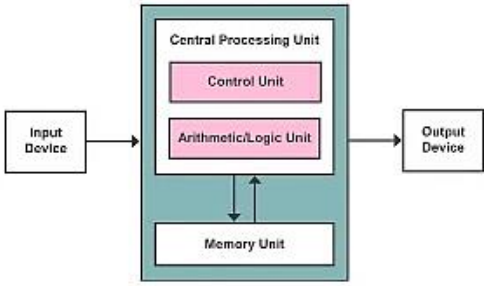
```
>>> help('math')
```

Loops

Here we will be looking at functions that allows for loops and iteration in Python programming language. We will look at the following 'if statements', while loop and for else loop.

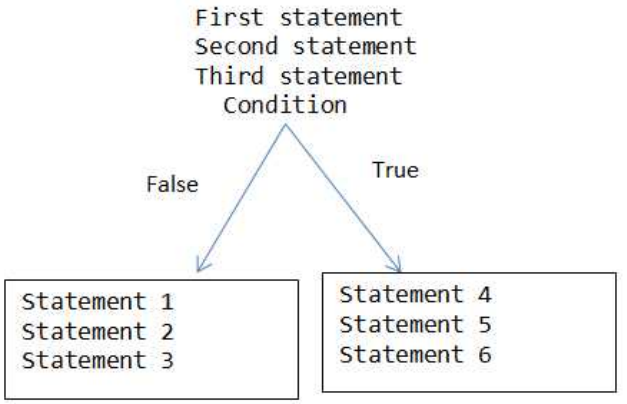
- if
- if else
- if elif else
- Nested if
- While loop
- For Else

Before we continue, we will talk about the Central Processing Unit (CPU) of a computer.



Like most of us know, CPU has three parts – the Memory Unit, Arithmetic /Logic Unit, and the Control Unit. Up to this point, we have been working with the arithmetic unit with which we have been performing calculation and the Memory Unit to save data, store variables, etc. But in this section, we will be working with the logical unit. This is the part of the CPU that allows your computer to think using different conditions. In programming, we often write conditional statements before

taking decisions. The general form for writing conditional statements is as follows;



If Statement

Whenever we use the IF conditional statement, the program only executes if the condition is true, else it will do something else. Note that IF statement is a block statement which is called 'Suite' in a Python programming language where you can write multiple statements. Note that indentation (some space) is used in IF conditional Statements to indicate statements or conditions that belong to the IF statement. By default, the 'tab' key on the keyboard produces four (4) spaces, so we usually use four (4) spaces for indentation for the IF Statement. Understand that any amount of space will work. Lets look at the general format of an If conditional Statement;

```
If Statement is True;
      Execute Condition
```

Example: Let's look at an IF statement where x = 3. If x is divided by any number and the remainder is 0, then x is an even number else x is an odd number. To execute this in Python, we have;

```
>>> x = 8
>>> r = x%2
>>> if r==0:
       print("Even Number")
>>> if(r==1):
      Print("Odd Number")
Even Number
>>>

>>> x = 7
>>> r = x%2
>>> if r==0:
      print("Even")
>>> if(r==1):
      Print("Odd")
Odd Number
>>>
```

As programmers, we would want to improve the performance of codes; it is not about writing codes and getting outputs, but it is about writing an efficient code. The above IF Statements are not efficient because when you say your number is eight (8) and we know eight is even. It will check if r is equal zero, in this case, r is zero (0), and it will still go ahead to check if r is equal one (1) before outputting "Even Number." To make the code efficient, it is not supposed to check if r is equal to one (1) after checking for r == 0 and

finding the condition to be true before outputting 'Even Number.' Hence, we will introduce the 'else' statement.

IF Else Statement

Instead of using two "IF", we use the 'if and else statement'. So the 'if statement' in general term will be written thus;

```
if Condition
      Output True
else:
      Output False
```

If the first condition is true, the program won't bother checking the else condition before outputting result. Always add a colon at the end of an else statement whenever you use an 'if else statement.'

```
>>> x = 8
>>> r = x % 2
>>> if r==0:
      print("Even Number")

else:
      Print("Odd Number")
Even Number
>>>
```

Nested If Statement

We can print an 'If Statement' within another 'If Statement' called 'nested if'. For instance, from our previous example we can add another if statement to output 'large number' if x is greater than 5.

```
>>> x = 8
>>> r = x % 2
>>> if r==0:
      print(“Even Number”)
      if x>5:
            print(“large number”)
      else:
            print(“not a large number”)
else:
      Print(“Odd Number”)
Even Number
Large number
>>>
```

Example:

```
>>> x = 3
>>> r = x % 2
>>> if r==0:
      print(“Even Number”)
      if x>5:
            print(“large number”)
      else:
            print(“not a large number”)
else:
      Print(“Odd Number”)
Odd Number
Not a large number
>>>
```

If elif else Statement

Lets say we have a program where if a user specifies 1 we will output 'one', when a user says 2 we will output 'two', when a user says 3 we will output 'three' and so on.

```
>>> x = 2
>>>If x == 2:
      Print (“two”)
```

```
>>>If x == 3:
      Print(“three”)
>>>If x==4:
      Print(“four”)
two
>>>

>>> x = 4
>>>If x == 2:
      Print (“two”)
>>>If x == 3:
      Print(“three”)
>>>If x==4:
      Print(“four”)
four
>>>
```

Note that before this code outputs any number, it goes through the entire if statements. Instead of allowing it to go through all the 'if statements' before outputting any value, we can use the elif statement. Elif means 'else if' which means only if a condition is not true before the next one can be checked. Here is how we do it;

```
>>> x = 2
>>>If x == 2:
      Print(“two”)
>>>elif  (x == 3):
      Print(“three”)
>>>elif x==4:
      Print(“four”)
two
>>>
```

Now if we have a value that does not match the given input values, you can then introduce an 'else statement' to the 'elif

statement.' Lets take for example, we assign 5 to x, and 5 is not in the 'elif conditions,' we will introduce the else statement which will output a string to address such condition. E.g.,

```
>>> x = 5
>>>If x == 2:
      Print("two")
>>>elif (x == 3):
      Print("three")
>>>elif x==4:
      Print("four")
else
      Print("value not found")
Value not found
>>>
```

For Else Statement

If you are coming from a different programming language like C, C++, and Java, you will be conversant with 'For Loop' and we know that 'else' can be used with the help of 'if' statement. You may have never encountered 'For' and 'else' together. However, in Python, we can use 'For' and 'else' together. To understand this, let's take an example.

Example: Let's say we have a list 12, 15, 18, 21, 26 and we want to check if this list has numbers which are divisible by 5. If there are numbers divisible by 5, then we will print them. So let's handle this problem using a 'for else loop.'

```
>>> nums =  [12,15,18,21,26]

>>> for num in nums:
      if num % 5 == 0:
```

```
      print(num)
15
>>>
```

In Line 3, the `if num % 5 == 0:` statement means, if num is divided by 5 without a remainder, then print num. From the list, 15 is the only number divisible by 5 without a remainder; hence the output is 15.

Example 2:

```
>>> nums = [12,15,18,20,26]

>>> for num in nums:
      if num % 5 == 0:
      print(num)
15
20
>>>
```

If we want the program to output only the first number that is divisible by 5 irrespective the numbers divisible by 5 in the list, then we introduce 'break'. The 'break' function terminated a loop.

```
>>> nums =  [12,15,18,20,26]

>>> for num in nums:
      if num % 5 == 0:
      print(num)
      break
15
>>>
```

```
>>> nums =  [10,15,18,20,26]

>>> for num in nums:
      if num % 5 == 0:
      print(num)
      break
10
>>>
```

Now let's add the 'else condition' to take care of situations where the list has no number divisible by 5 without a remainder.

Example:

```
>>> nums =  [11,13,18,21,26]

>>> for num in nums:
      if num % 5 == 0:
            print(num)
            break
else:
      print("not found")
not found
>>>

>>> nums =  [10,13,18,21,26]

>>> for num in nums:
      if num % 5 == 0:
            print(num)
            break
else:
      print("not found")
10
>>>
```

Example: Let's use the 'For else' statement to write a code which will detect if a number is prime or not. Let's take 7 as our example

```
num = 7
>>> for i in range  (2,num):
      if num % i == 0:
            print("Not Prime")
            break
else:
      print("Number is prime")
Number is prime
>>>

num = 10
>>> for i in range  (2,num):
      if num % i == 0:
            print("Not prime")
            break
else:
      print("Number is prime")
Not prime
>>>
```

While Loop

In programming, loop means repetition of a statement. For instance, you have five statements and you intend to repeat them. You can either copy-paste the same code multiple times or use the 'while loop.'

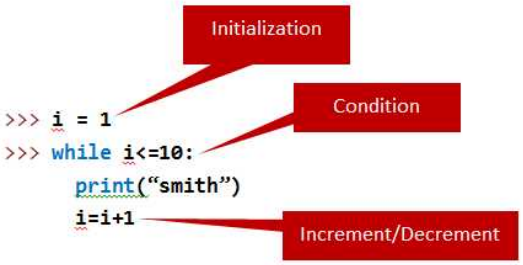
Example: Let's say we want to output the word 'smith' three times.

```
print("smith")
print("smith")
```

```
print("smith")
smith
smith
smith
```

Now, if I intend to print "smith" ten times, I will need to copy-paste the code ten times. But what if I want to print it ten thousand times? This becomes cumbersome. The best way to print multiple times is to apply a loop. We can do this using the 'while' or 'for' loop, but we will focus on 'while loop' here.

Example:



Note that the colon **(:)** in line two indicates, "whatever is coming after it is part of the block. That is;

```
print("smith")
    i=i+1
```

is part of the `while i<=10:` block and

```
print("smith")
    i=i+1
```

belongs to the same suite.

So when we run the code, we will have smith ten times.

```
>>> i = 1
```

```
>>> while i<=10:
      print(“smith”)
      i=i+1
smith
smith
smith
smith
smith
smith
smith
smith
smith
smith
>>>
```

We can as well decrement the while loop.

Example:

```
>>> i = 10
>>> while i<=10:
      print(“smith”, i)
      i=i-1
smith 10
smith 9
smith 8
smith 7
smith 6
smith 5
smith 4
smith 3
smith 2
```

```
smith 1
>>>
```

Nested While Loop

We can use a 'while loop' within another 'while loop' and this is referred to as 'nested while loop.' In 'nested while loop,' you first execute the inner loop before you attend to the outer loop.

Example: Let's print two different strings (smith and kelvin). We can print 'smith' once and kelvin five times. Here is the code to do that;

```
>>> i = 1
>>> while i<=5:
      print("smith")
      >>> j = 1
      while j<=4:
            print("kelvin")
            j=j+1
    i=i+1
smith
kelvin
kelvin
kelvin
kelvin
smith
kelvin
kelvin
kelvin
kelvin
```

```
smith
kelvin
kelvin
kelvin
kelvin
smith
kelvin
kelvin
kelvin
kelvin
smith
kelvin
kelvin
kelvin
kelvin
>>>
```

If we want the outputs to be printed on the same line, then we introduce end="" to each print statement. For example;

```
>>> i = 1
>>> while i<=5:
      print("smith ",end="")
>>>   j=1
      while j<=4:
            print("kelvin ",end=""))
            j=j+1
    i=i+1
    print()
smith kelvin kelvin kelvin kelvin
```

```
smith kelvin kelvin kelvin kelvin
smith kelvin kelvin kelvin kelvin
smith kelvin kelvin kelvin kelvin
smith kelvin kelvin kelvin kelvin
>>>
```

Python as a Scripting Language

Programming languages are compiled while scripting languages are interpreted. Python is both a programming language and a scripting language which means scripts can be written to automate a certain task. Scripts are less code-intensive as compared to most traditional coding language because they do not require the compilation steps and are rather interpreted. They are the building blocks for larger Raspberry Pi projects. In this section, we will look at an example of a script that you can write, call it on the Raspberry Pi, and a walk through each of the functions in the script for a better understanding.

An Example of a script

```
print('This is my first script') # this is a comment
string1 = 'meet me'
string2 = 'this afternoon'

# strings can be concatenated with + and repeated with *
print(string1 + '  ' + string2 + '!'* 3)

twothird = 2/3
print('2 divided by 3 is about {}'  .format(twothird)) # you
can refer to a variable input
print('or with fewer decimal points: {:f}'  .format(twothird))
# you can explicitly format the output
```

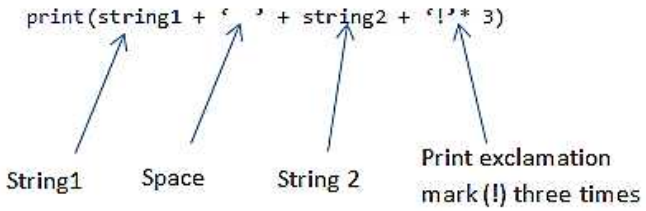
```
import math #usually we put imports at the start, I put it
here for context.
String3 =  'more'
print('you can print {0} than one variable,here\'s pi:  {1}.
That makes {2} variable!'  .format(string3,math.pi,3))

print('\n') # a blank line
for num in range(2,10,2): #count in twos, if you like.
       Print(num)

print('\n') # a blank line
a = [1,2,3,4,5,6,7,6,15,4,3,2,1] # This is an array

# Here's a FOR loop with an IF statement inside!
for element in a:
       If element == max(a): #looks for the largest number in
a
       print(element*'=')
       print('We reached the largest')
print(element*'=')
```

On the python shell, click on "File" - "New File" then type the script in the new window. After writing the entire script, ensure you save it with a .py extension. For this example, you can save it as "firstscript.py" and then run it. On the Menu Click on "Run" - "Run Module" or press "F5" on the keyboard. When the script compiles, we have the following on the screen.

```
This is my first script
Meet me this afternoon!!!
2 divided by 3 is about 0.66666666666666666
Or with fewer decimal points:  0.666667
You can print more than one variable, here's pi:
3.141592653589793. That makes 3 variables!

2
4
6
8
```

```
=
==
===
====
=====
======
=======
We reached the largest
=======
======
=====
====
==
=
```

Note: # is used for comments. This is to improve the readability of a code. By putting a comment, we can leave a message or messages for ourselves or other programmers about what the code is doing in a more human readable form. In line 2 we created two string variables called 'Meet me' and 'this afternoon'. In line 5, we saw the function;



It is called string concatenation (when two separate things are brought together). The asterisk (*) is a repeat symbol. Which means print **'!'** three times. The result of the above concatenation function is;

Meet me this afternoon!!!

In line 6, we created a variable called `twothird = 2/3` and in the next couple of lines we are printing some variables with the print statement. So we can print on the screen two divided by three is about `print('2 divided by 3 is about {}'.` The curly bracket at the end of the statement allows information to be inserted into the print statement. We do that with the .format argument. In .format we use the variable twothird. What this means is that the variable 'twothird' will be substituted for the curls braces { }. The same thing happened on the next line except that within the curly braces we can include formatting information `{:f}`. In this case, we are formatting the output as a float. A float is a decimal representation of numeric data. The default format for float is six decimal places. In line 10, is the 'import statement `(import math)`. The import statement is used to import functionality; in this case, we are importing a mathematics toolbox. Math has several routines and constants to use. By convention, we put 'import' at the top of the script, but if this is done, you will not be able to have commands above that import statement. In line 13, we created a string called 'more.' In the next line, we have a print statement `print('you can print {0} than one variable, here\'s pi: {1}. That makes {2} variable!' .format (string3,math.pi,3)` with three variables [{0}, {1} and {2}] being displayed (where 0 represents string3, 1 represents math.pi, and 2 represents the number 3) and by including an index inside the curly braces, we can explicitly pick and

choose which variables we are pulling out of that format argument. Swapping the indexes in the curly braces automatically swap the variables you drew. The output of this print statement is, **"You can print more than one variable, here's pi: 3.141592653589793. That makes 3 variables!". Maths.pi** invokes Pi constant from the maths package. The **'\n'** is a special escape sequence used for line breaks or creates a new line.

In line 17, we have a loop, it is a For loop;

```
    for num in range(2,10,2): #count in print(num)
                                twos, if you like.
```

Indentation

What this loop means is that we will make up for some variables num in range two to ten incrementing by 2 (2,10,2). In other words, we will make up some variables called 'num,' which should take on the values in the range two to ten but should be incremented by two each time. Note that the print (num) is indented. Python uses that wide indentation space for code structure. If the indent is not present, it will return an empty 'for loop'. The print (num) returns

2
4
6
8

It doesn't return 10 because as the loop increment from 8 - 10, it got to the end of its range, preventingit from exceeding the set loop condition.
We print another blank line after writing the for loop with the function;

```
print('\n') # a blank line
```

then we create an array,

```
a = [1,2,3,4,5,6,7,6,5,4,3,2,1]
```

An array is a list which doesn't necessarily hold numbers. It could hold strings etc. The array provided will be used in the second loop, which is different style of invoking a 'for loop.' Here we say,

```
for element in a:
    If element == max(a): #looks for the
                          largest number in
                          a.
```

Where 'a' is an array. This is saying, for everything in 'a', run some code and then increment to the next one. The 'If statement' is used to branch our logic or make decisions for the 'for statement.' The 'if' statement is interpreted thus; if the current element we are looking at is equal to the maximum value in that array (in this case 7), then do this;

```
print(element*'=')
print('We reached the largest')
```

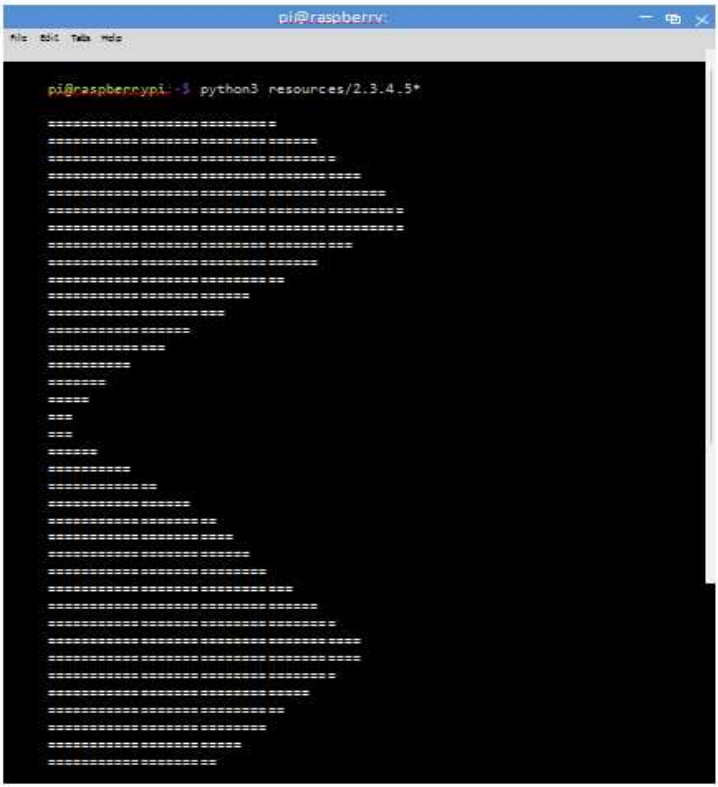
otherwise, do this;

```
print(element*'=')
```

What this statement does is to print a set of 'equal signs (=)' incrementally until it gets to seven (7) which is the maximum value in the array. When it gets to seven, you will get an output with the word 'we reached the largest.' Then the 'equal sign' should be printed in decreasing order until it gets to the minimum value in the array (which is 1).

Math and Functions Scripts for Oscillating Sine wave

The script we will arrive at will produce a nice animation in the prompt or shell of a sine wave. You can run a Python script from within the console window by typing $ python3 followed by the path you kept your file. For instance, if the script is stored in a directory called resources/2.3.4.5; we will access the script via `pi@raspberrypi:-$ python3 resources/2.3.4.5`* and the python script will be executed when you hit enter on the keyboard. The script we are about to write will animate a series of sine waves.



Now, let's achieve this effect from scratch.

- Open the Python shell
- Click on "File" – "New File"
- Click on "Save As" and select the directory you wish to save it. Mine is 2.3.4.5.py

To get the effect working, we will concatenate strings again. In the first script we tried, we had the concatenation of equal signs. We will be doing the same thing here, but the differ-

ence is that we will be using a little bit of math to calculate how many equal signs we will use and also define the number of cycles we want the sine waves to go through;

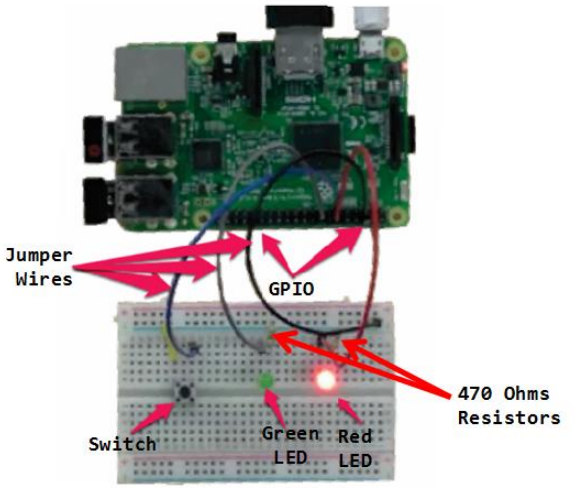
```
Import math
Import time

numofCycle = 6
pi = math.pi

def    sin(x):
       return math.sin(x)

x = 0
while x <  (numofCycle * 2 * pi):
       bar = int(20*sin(x)) # length of equal sign
       x += 0.3   # same as x = x+0.3
       print((21+bar)* '='))
       time.sleep(0.03) #slows down the animation
```

You can now run the program

Chapter Four

Connecting Raspberry Pi to External Devices

Now that we are familiar with Python programming, we are ready to connect our Raspberry Pi to external devices via the GPIO and write Python codes to control them using the Raspberry Pi. In this chapter, we will be carrying out some simple projects such as simple button input and LED output circuit. Always remember to power OFF your pi when you prototype circuits to prevent bridging that could damage your Pi. Double-check your circuits before powering it ON.



To carry out this project, we need the following items

- Breadboard
- 470 ohms resistors for each LED
- Two LEDs (One Green and One red)
- Button switch

- Some male-to-female jumper wires
- Breadboarding wire

The intention of this circuit is to write a program that would vary the intensity of the brightness of the red LED (i.e. it dims and brightens) while the green one will blink when we press the switch. To achieve this, we will create a new file from Python IDLE and click save as (e.g. 2.3.4 LEDproject.py) to save the code in a directory. In this project, it is important to import the Raspberry GPIO (line 1). This is a specific package for the Raspberry Pi to control the GPIO. It takes in all the functionality of the GPIO. In line 2, we imported time; this gives us some delays that are used to create a blinking effect on the LED. It is always a good idea to define pins at the beginning of the script. These are labeled "pin numbers" chosen for the project. For this project, the LED that is being PWMed (the blue led) is on pin 18. PWM is the effect that allows you to control the brightness of your LED. The LED that will be blinking (the red Led) named redledPin is on pin 23 and the button switch which we defined as butswtPin is on Pin 17. We also need to define the duty cycle for this circuit. For this, we will define a variable called dutyCyc and assign 80% to it. A duty cycle defines how much power the pin will output and this will determine the brightness of the LED. With PWM, the duty cycle is between 0% - 100%. Note that you can change the circuit and still have the original functionality by changing the definitions for the pins in the code and everything will run just fine. After, defining the duty

cycle on our code, we will setup the GPIO using the GPIO label we used initially. First we will set up the GPIO mode to `GPIO.setmode(GPIO.BCM).` This is essentially setting the GPIO numbering as the one defined by the manufacturer of the chip. That is to use the 'pin numbers' we have defined effectively; we need to use the Broad Cam (BCM) numbering system. Next, we will write a code to define the functions of each pin we will be using – that is whether they should act as INPUTS, OUTPUTS, OUTPUTS with PWM. In line 8 on our code, we will set up the pins, startingvwith the redledPin as output (`GPIO.setup(redledPin.GPIO.OUT).`

On the next line, we will also set pwmPin as output `GPIO.setup(pwmPin.GPIO.OUT),` the button switch (butswtPin) will be set to GPIO IN `(GPIO.setup (butswtPin.GPIO.IN)` because we are using the button as an input. When we use buttons as input without any other supporting hardware, we do that by activating what is called an internal pull-up resistor, and the schematic for the internal pull-up resistor for this project is `pull_up_down=GPIO.PUD_UP`. This code activeates the internal pull-up resistor which will allow us to use our button. After defining the function of each pin, we will then initialize the PWM channel. We have set up the PWMPin as an output, but we haven't attached it to any PWM properties. So we will create an object called pwm and invoke GPIO.pwm to attach it to pwm functionality by choosing the pwmPin and attaching a frequency by which it will run. Here we will use 200Hz. So we have `pwm = GPIO.pwm(pwmPin,`

`200).` The next piece of code `(GPIO.output(redledPin .GPIO.LOW)` will set our blinking LED OFF initially and the code `pwm.start(duty)` will start our pwm channel with a duty cycle of 80%, which we have defined initially.

After setting up the GPIO, in the next line, we will write a set of codes to enhance the functionality of the program. In this section, we will first write the program to set itself up and then loop the actual functionality after its been set up using the while loop. The 'while loop' will continuously execute the program whenever the condition is true. In the 'while loop' we will set up a condition for the red LED to be brightened when the switch is not pressed and blink the green. While it slowly brighten and dims red LED when switch button is pressed. So if the button is not pressed we have the if statement below becomes true;

```
if GPIO.input(butswtPin):
    pwm.ChangeDutyCycle(dutyCyc)
    GPIO.output(redledPin.GPIO.LOW)
```

Else, if the button is pressed (`pwm.ChangeDutyCycle(duty Cyc)` make the red LED bright with the following code `(GP-IO.output(redledPin.GPIO.HIGH)).` Add a delay (in seconds) to have a blinking effect with the `time.sleep` function. Here we will set the delay time to 0.6 seconds. Then we will dim the red LED with the code lines

```
pwm.ChangeDutyCycle(100-dutyCyc)
GPIO.output(redledPin.GPIO.LOW)
time.sleep (0.6)
```

Note that the 100 – dutyCyc subtracts the defined duty cycle which is 80 from 100. Therefore the new duty cycle for the red LED is 20, which dims it.

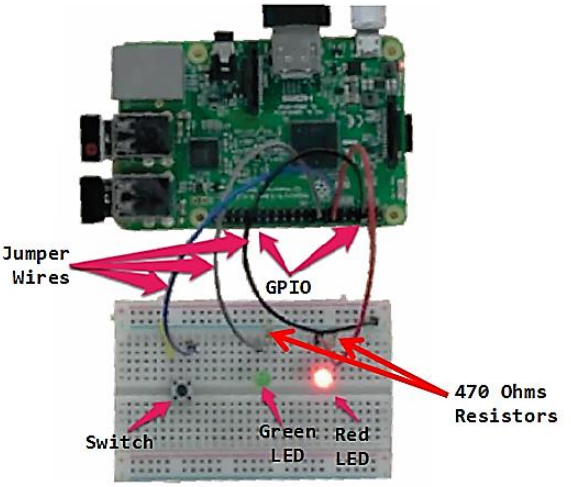
Finally, we will wrap our entire script in a "try." When we write a script that initializes GPIO like in the setup below, this sets up the pins to carry out the defined functions like light the LED, dim the LED etc, but when we quit the script, we leave the pins in that defined state. We have to clean this up GPIO setup when we quit the script, and we do this by wrapping our entire script in what is called 'try.' The part we wrap in the script, in this case, is the 'While loop.' To do this,

Highlight the content of the 'while loop' – click on 'Format' – select 'indent region' – or click on 'ctrl +]' – this highlights the while loop, then you can put the entire loop within the 'try' function and the exception to 'try' function (`except: keyboardInterrupt:`). The code will try whatever is within the 'while loop' and when keyboardInterrupt happens, it will run the exception to the 'try.' The 'keyboard interrupt' is how we quit our script and in that 'except' part of the code is where we reset the entire GPIO to a safe place so we can use the Raspberry Pi for other things. Within the 'except,' we will invoke `pwm.stop()` which cuts the pwm channel, and then we run `GPIO.cleanup()` which will allow us to quit the program and have all the GPIO pins reset to a safe state. Now let's look at the script we have been discussing below.

```
Import RPI.GPIO as GPIO # this controls the GPIO
Import time # produces delays for blinking effect

#pin definition
pwmPin = 18
redledPin = 23
butswtPin = 17

dutyCyc = 80

#GPIO Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(redledPin.GPIO.OUT)
GPIO.setup(pwmPin.GPIO.OUT)
GPIO.setup(butswtPin.GPIO.IN,  pull_up_down=GPIO.PUD_UP)
pwm = GPIO.pwm(pwmPin, 200)

GPIO.output(redledPin.GPIO.LOW)
pwm.start(duty)

try:
       while 1:
              if GPIO.input(butswtPin):
              pwm.ChangeDutyCycle(dutyCyc)
              GPIO.output(redledPin.GPIO.LOW)

       else:
              pwm.ChangeDutyCycle(dutyCyc)
              GPIO.output(redledPin.GPIO.HIGH)
              time.sleep  (0.6)
              pwm.ChangeDutyCycle(100-dutyCyc)
              GPIO.output(redledPin.GPIO.LOW)
              time.sleep  (0.6)
except:  keyboardInterrupt:
       pwm.stop()
       GPIO.cleanup()
```

Finally, click on 'run' – 'run module' if there are no error messages, then hold down the switch button to see the effect of the code on the LEDs.

Script to Create a Breathing LED Effect



In this project, we will write a script that causes a LED to breathe slowly, and when we hold down the switch, it increases the LED pulse rate. We will use all the hardware as in our previous project, but the code for the script will have some changes. We will copy the pin definitions and the GPIO setup of the previous script. The values of PWM will be changed to drive the LED to breathe smoothly in this project, giving us a duty cycle of between 0 – 100. We will write a function that will allow us to do this scaling which we will call "ScalFun." We will define ScalFun and parse a number we

intend to scale into it. The input has to exist between two numbers.

```
def ScalFun(x, inputLow, inputHi, outLow, outHi):
```

Where the x – is the number we intend to scale
inputLow – Lowest expected number on the scale
inputHi - the Highest expected number on the scale
outLow – Output for the expected lowest number on the scale
outHi - Output for an expected Highest number on the scale

Next, we will create two ranges of the number to expect. One for the input (inRange) and the other for output (outrange). The expression for input Range (inRange) connotes the difference between the lowest input value and the highest input value on the scale that is, subtract the lowest expected input value from the highest expected input value (inputHi – inputLow). The expression of the output Range (outrange) connotes the difference between the lowest ouput value and the highest output value on the scale (outHi – outLow).
With the input and output Ranges declared, we will need to express the input as a percentage. We can express that as

```
inScale = (x – inputLow)/inRange
```

What the above function does is normalizing x into inScale. If x is for instance zero (0), which is half way between negative one (-1) and one (1), inScale is saying that is 50% of that possible range. That is come up by 50% from negative 1 (-1)

to Zero (0), and this process is called normalizing. Once we have normalized the values (which tends into a number between zero (0) and one (1)), we then add it to the output (outLow) and multiply it by the output Range (outrange) using the function;

```
return outLow + (inScale * outrange)
```

Note, we are incrementing the x value, so we set x to zero (0)

We will decide how fast the LED breathes whether we press the button or not in the 'while loop.' For the 'while statement, we will say, if the pressing of the button did not occur, we want the breathing effect to go slowly.

```
while 1:
if GPIO.input(butswtPin):
        step = 0.05
```

Step here can be any number. Else,

```
step = 0.1
```

What the 'if statement' does is to change the variables when we press the button. In other words, it is choosing how big the 'steps' should be, and this will directly affect how fast the LED is breathes. After we are done with the while loop, we set our duty cycle using the scale function (scalFun).

```
dutyCyc = scalFun (sin(x),-1,1,0,100)
```

We know that sin x returns values between negative one (-1) and one (1), and we want to rescale this to a duty cycle

(dutyCyc) between zero (0) and one hundred (100). With one call to our special scale function (scalFun), we have been able to go from a sine wave in one domain to a duty cycle compatible with how to drive our LED. After setting up the duty cycle, we can bring in our 'step' by saying;

```
X += step
```

Finally, we set out pause (`time.sleep`) to 0.06 seconds. Now let's look at the script we have been discussing below;

```
Import RPI.GPIO as GPIO # this controls the GPIO
Import math
Import time # produces delays for blinking effect

#pin definition
pwmPin = 18
redledPin = 23
butswtPin = 17

dutyCyc = 0

#GPIO Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(redledPin.GPIO.OUT)
GPIO.setup(pwmPin.GPIO.OUT)
GPIO.setup(butswtPin.GPIO.IN, pull_up_down=GPIO.PUD_UP)
pwm = GPIO.pwm(pwmPin, 200)

GPIO.output(redledPin.GPIO.LOW)
pwm.start(duty)

def ScalFun(x, inputLow, inputHi, outLow, outHi):
       inRange = inputHi - inputLow
       outRange = outHi – outLow
       inScale = (x – inputLow)/inRange  #normalizing
```

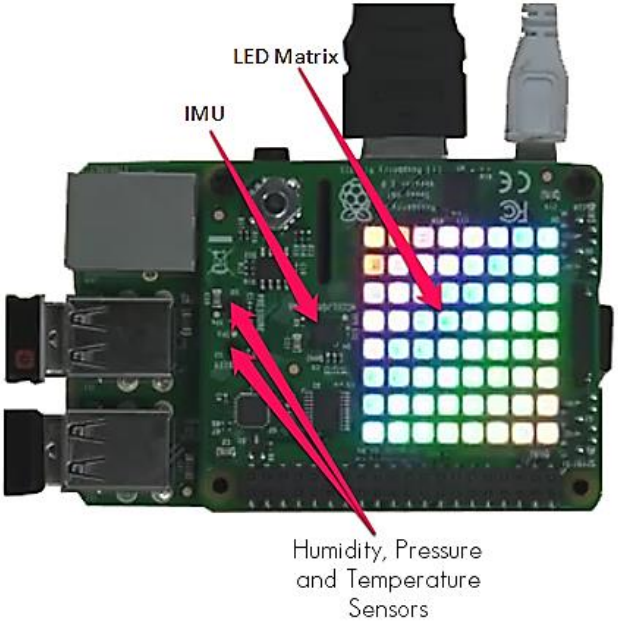
```
        return outLow + (inScale * outrange)
x = 0
try:
        while 1:
                if GPIO.input(butswtPin):
                step = 0.1
        else:
                step = 0.3

         dutyCyc = scalFun(math.sin(x),-1,1,0,100)
           x += step
           pwm.ChangeDutyCycle(dutyCyc)
           time.sleep(0.06)

except: keyboardInterrupt:
        pwm.stop()
        GPIO.cleanup()
```

Finally, save your script and run it.

HATS

HATS which simply means hardware attached on top describes a hardware module that one can plug on top of the Raspberry Pi connecting via the GPIO. This makes it quick and easy to connect your Pi to different pieces of hardware with which to experiment. An example of a HAT is the sense HAT. The sense Hat is used for several Raspberry Pi projects due to its onboard LED Matrix, International Measurement Unit (IMU), and several inbuilt sensors. The LED matrix displays data, color code, etc. The IMU consists of a gyroscope, an accelerometer, and a magnetometer which enables the sense HAT to measure movements, altitude, magnetic field, etc. The sense HAT is used also as a weather station because

it consists of several built-in humidity, pressure, and temperature sensors.



Sense HAT

Scrolling Rainbow Effect on the Sense HAT

We can now write a script that gives the LED matrix of a sense hat a scrolling rainbow color effect. Firstly, attach the sense hat to the Raspberry Pi - power on the Pi and open a new file.

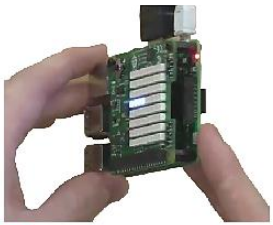
```
from colorsys import hsv_to_rgb
from time import sleep
from sense_hat import SenseHat
```

```
#  Hues represents  the spectrum  of  colors  as values
#  between 0  and  1.  The  range is  circular so  0
# represents red, -0.2 is yellow, -0,33 is green, 0.5 is
# cyan, -0.66 is blue, -0.84 is purple and 1.0 is back #
to red. These are the initial hues for each pixel in the
display.

Hues = (
       0.00, 0.00, 0.06, 0.13, 0.20, 0.27, 0.34, 0.41,
       0.00, 0.06, 0.13, 0.21, 0.28, 0.35, 0.42, 0.49,
       0.07, 0.14, 0.21, 0.28, 0.35, 0.42, 0.50, 0.57,
       0.15, 0.22, 0.29, 0.36, 0.43, 0.50, 0.57, 0.64,
       0.22, 0.29, 0.36, 0.44, 0.51, 0.58, 0.65, 0.72,
       0.30, 0.37, 0.44, 0.51, 0.58, 0.66, 0.73, 0.80,
       0.38, 0.45, 0.52, 0.59, 0.66, 0.73, 0.80, 0.87,
       0.45, 0.52, 0.60, 0.67, 0.74, 0.81, 0.88, 0.95,
       )

Hat = Sensehat()

def scale(v):
    return int(v * 255) # controls LED brightness

while True:
     # creates the scrolling effect of the hues
     hues =  (h * 0.01) % 1.0 for h in hues)
     # coverts the hues to RGB values
     pixels =  (hsv_to_rgb(h, 1.0, 1.0) for h in hues)
    # hsv_to_rgb returns 0..1 floates: convert to ints
      in the range 0..255
    pixels = ((scale (r), scale(g), scale(b)) for r,
    g, b in pixels)
    # Update the display
    hat.set_pixels(pixels)
   sleep(0.04)
```

Finally, Save and run the code.

Plum Line Effect on the Sense HAT

The Plum line effect is a script that allows the light on the LED matrix of a Sense HAT to stretch to the side of the raspberry pi titled towards the ground. When the Raspberry Pi is placed horizontally, the light stays at the middle.



2 Tilting the Sense HAT to the Left moves the LED Light to the Left

1 Placing the Sense HAT Horizontally keeps the LED Light at the Center

3 Tilting the Sense HAT to the Right moves the LED Light to the Right

4 Tilting the Sense HAT downwards moves the LED Light towards the ground

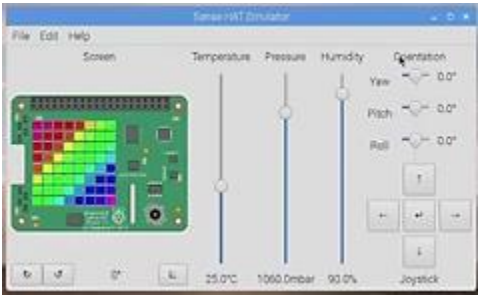
```
from time import sleep
from sense_hat import SenseHat
from PIL import Image, ImageDraw

hat = SenseHat()

hat.clear()
origin =  (7, 7) # position of light when pi is level
while True:
       a = hat.get_accelerometer_raw()
       img = image.new('RGB', (15, 15))
       draw = ImageDraw,Draw(img)
       dest =  (origin(0) * a['x'] * 7.0, origin[1] *
       a['y'] * 7.0)
```

```
draw.line([origin, dest], fill*(0, 0, 255),
width*3)
img = img.resize((8, 8), Image.BILINEAR)
hat.set_pixels(list(img.getdata()))
sleep(0.04)
```

Notes:
dest – destination
fill in (fill*(0, 0, 255),) means fill color. We can brighten the light of the LED by changing the values of the fill color to fill*(255, 255, 255).

The Sense Hat Emulator

In the latest Raspbian plus pixel operating system, we have in our programming menu, the sense hat emulator. To access this emulator, on the Raspbian menu, click on 'Programming' – 'Sense HAT Emulator.'



The Sense HAT emulator allows us to run codes for the sense HAT even if we don't have the physical sense HAT. You can still see what effects it will produce as well as play around with the inputs on the prototype. You can tilt the sense HAT emulator by changing the orientation values. You can as well adjust temperature, pressure and humidity of the sense HAT on the emulator using the temperature, pressure and humidity

slider. The use of the joystick is for navigating menus. To allow our codes to run on the sense HAT emulator rather than the physical hardware, in the script, change the code line;

```
from sense_hat import SenseHat
```

to

```
from sense_emu import SenseHat
```

and then save and run the script. You can make the code run on the physical sense HAT by replacing the "_emu' with "_hat."

Chapter Five

Shell Scripting

A shell is a program that takes written commands and processes them. A shell script is the sequence of these commands, and they are powerful tools which allow us to automate otherwise tedious tasks such as stacking programs as the Pi boots or run scheduled programs. Several interpreters that can be used to interpret shell scripts, but we will be using the 'bash interpreter' for our projects. So the first line of our script is used to tell our Raspberry Pi what interpreter should be used to interpret our commands and we do that by writing the command;

```
#!/bin/bash
```

The `\n` command allows us to create a new line, and `\t` creates a tab on a command line. Another very important command that you should take note of is the `echo` command. The echo command is a command that prints text to the prompt. This is a command that gives us feedback from our program, while we use the dollar ($) symbol to access variables or arguments. We use the ($0) to access the content of the argument with index 0. While we use the ($1) to output the first argument we entered. One more thing you need to note is the $#. The $# symbol does not access any argument, in particular, it accesses how many arguments where parsed.

For instance, if we want to output the word "the command entered was" we use the echo command

```
echo The command entered was
```

But if we intend to add an argument or a file path to an argument, we add the dollar symbol and zero at the end of the echo command line.

```
echo The command entered was $0
```

If we parsed information into the script (e.g. names of students) and we want to output the first name in the argument, we can say

```
Echo my name is $1
```

As the number increases, it accesses the next arguments `($0,$1,$2,$3.....$n).` If the `\n` appears at the end of an echo command creates a new line without issuing a new echo command. `We can also echo and parse in the option -e.` This allows us to format our output. An echo parsed in the -e option should be placed in a quotation mark. For instance, we can format our output to start on a new line, have a paragraph (tab), etc.

```
echo –e "His name is $2\n\t$# argument were passed in"

echo –e "I am $2 years old\n\t$# argument were passed"
```

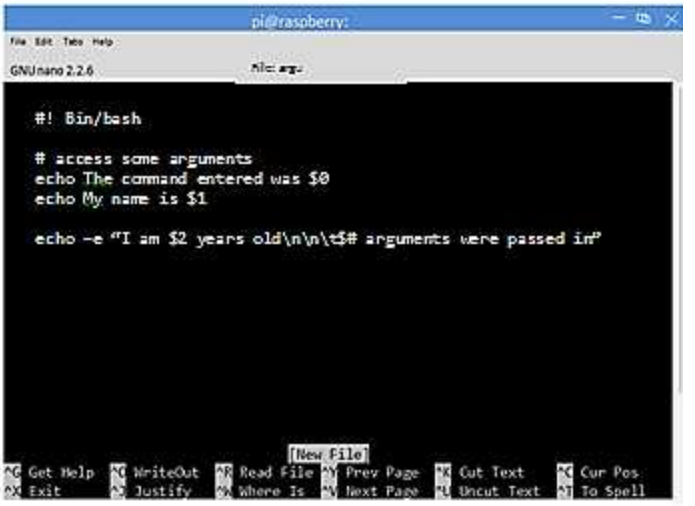
We use ctrl S to save shell scripts with any name and 'Y' to say "Yes" to the prompt to save.

File Permissions and Arguments

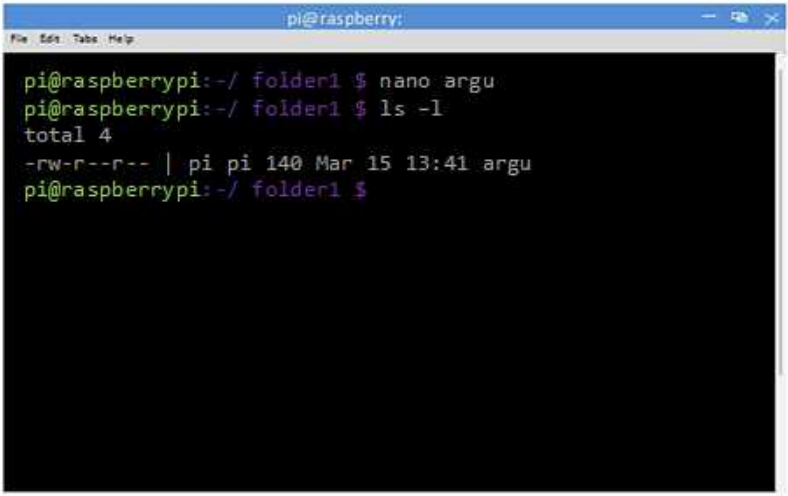
In this section, we will write a script that will take some arguments and print some texts with those arguments. If you remember from Python in chapter 2, arguments are pieces of information that we parse into a program or function. To start, we go to our Raspbian home screen (desktop), and we create a 'folder1' directory by right-clicking on the desktop – click on 'create new' – 'folder'. You can do this graphically or through the shell. We can now open the 'folder1' directory in the shell to create a script which we will called 'argu' (for arguments) using the command;

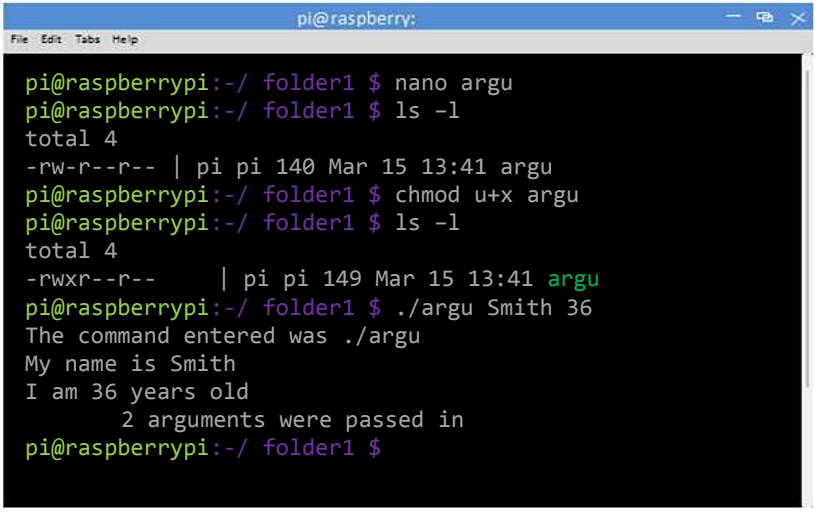
```
pi@raspberrypi:-$ cd folder1
pi@raspberrypi:-/ folder1 $ nano argu
```

We will parse some arguments into this script. This opens up the nano file editor and we write the command below;



Save the file and exit the nano file editor. Run the 'list command.' This will display details of the file and the permissions, (r-read, w-write, x-execute permissions).





We can change file permissions in groups of threes **rw-, r--** and **r--.** The groups of threes include the **file owner (rw-) the owner's user group**, if there are multiple users (r--) and **everyone else (r--).** You can modify any of these permissions using the **'chmod command.'** For instance, let's modify the file owner (user) file permission to include 'execute' permission.

```
chmod u+x argu
```

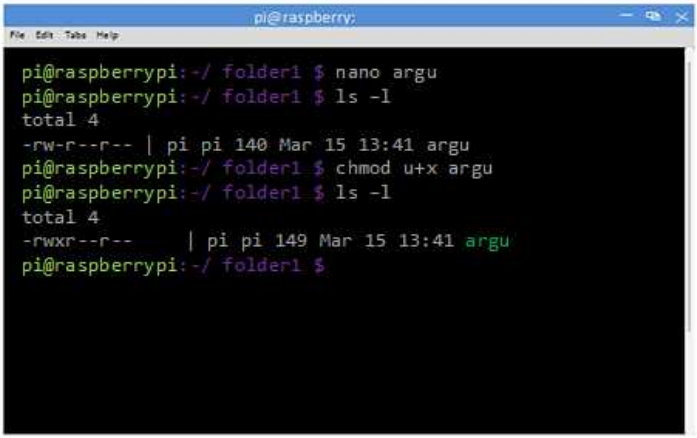
Where `U stands for the User's file`

`+ stands for add`

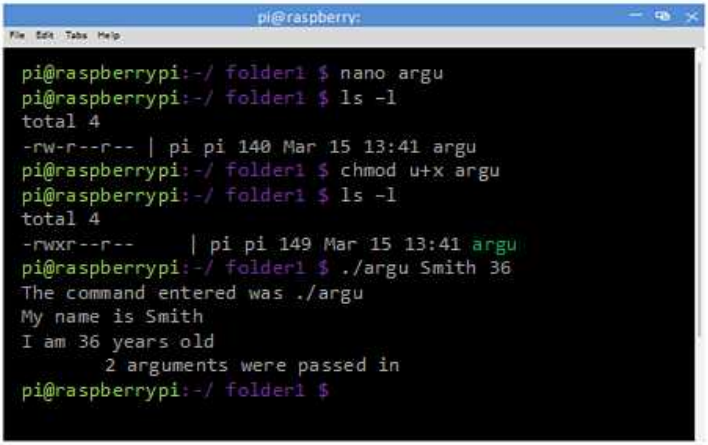
`x stands for execute`

`argu stands for the file we want the modification on`

Now, what this means is that we want to add an 'execute' permission to the user's (owner) file. When we run the 'list command' again, the details of our shell file will change with the user's file having an 'execute' permission and the color of the file name 'argu' has change to indicate that the file is now executable.



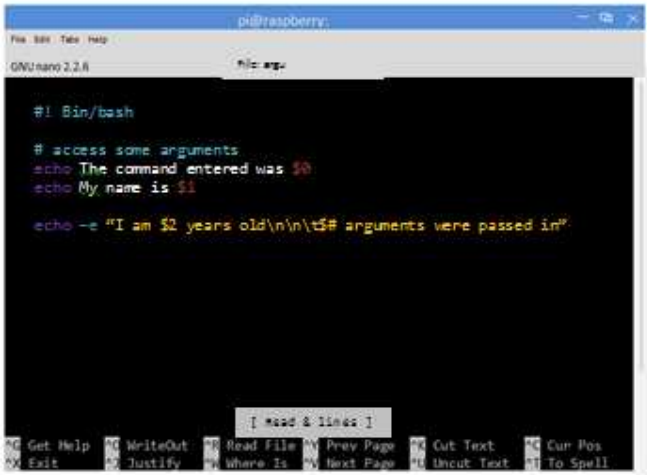
We are now ready to run our script. To run the script, we start up with a dot and a slash and the name of the file (./ argu). What this means is that, in this directory, there is a script called 'argu.' We passed two arguments in the script – first been 'My name' and the second is 'my age' then we press the enter key on the keyboard to display the output.



On line 13, we have a new line and a tab which is as a result of the text formatting that we did on the script viz

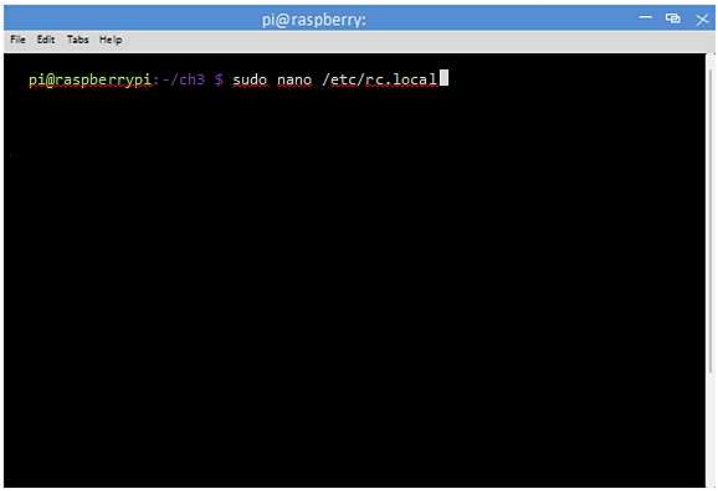
```
echo –e "I am $2 years old\n\t$# argument were passed"
```

If we run 'nano' again, you will discover the script is now color-coded as shown below;



This is called synthax highlighting. It is nano's way of showing us the meaning of the different keywords.

Scheduling Tasks and Running at Startup



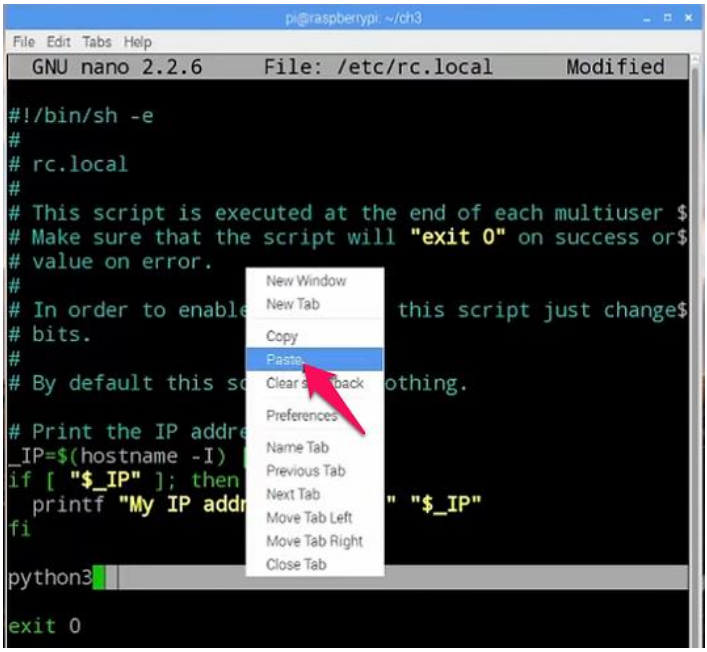
We will look at how to run programs as the Raspberry Pi is booting and also how to schedule programs to run periodically. You may want to run programs at startup if they are programs you need to run immediately. For instance, you may want certain applications to run at start-up, e.g., web servers, etc. To do this, we need to write a text file similar to the ones we have been running, and we need superuser permission to edit these files because they are system files. First, we will run sudo nano and then type the path to the file (/etc/rc.local).

When you click the enter key on the keyboard, the nano editor with a shell script opens up with a system file.

The system shell script is not quite similar to what we have been writing previously. On the script, we have 'hash' (#) by

each line. Also present, is an "if statement" which prints the Raspberry Pi IP address if it has one.

Commands that should run at start up are written after the 'If Statement", just before the exit 0 statement. We have to enter them before the exit 0 line otherwise; they won't run because the script will exit. For example, we can run the breathing LED script from chapter four to startup as the Raspberry Pi boots. To do this, we will run a command to run a Python script, then go to the folder that holds the breathing LED script – right click on the script and "copy path(s)" – then place our cursor in front of the python script command, right-click and select "paste." This pastes the absolute file path.



GNU nano 2.2.6 File: /etc/rc.local

```
#!/bin/sh  -e
#
# rc.local
#
# This script is executed at the end of each multiuser$
# Make sure that the script will “exit 0” on success $
# or value on error.
#
# In order to enable or disable this script just$
# change bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
If [  “$_IP”  ]; then
```

```
  Printf “My IP address is %s\n” “$_IP”
fi

python3/home/pi/2.3.4 LEDproject.py &

exit 0
```

Remember the breathing LED script has an infinite loop and the line of any script or program that has an infinite loop that is to be made to run at startup should have an epicenter symbol (&). The epicenter symbol must be at the end of the line, after a space; otherwise the Raspberry Pi won't boot; instead, it will get stuck in that loop. So you need to be very careful when you declare script(s) to run from c:/local . For instance, the breathing LED script when imported into our shell script should have a space followed by an epicenter (&) symbol at the end.

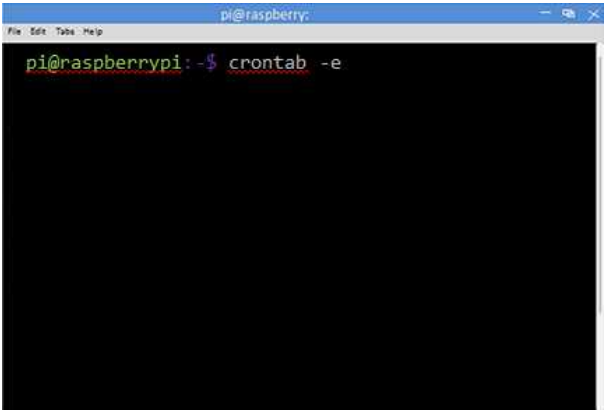
```
python3/home/pi/2.3.4 LEDproject.py &
```

When you finish, click on Ctrl S – to save, Y – for Yes and strike the enter key on the keyboard. With this, the next time you are rebooting the Raspberry Pi, if it is connected to an LED correctly, the LED will have that breathing effect at startup.

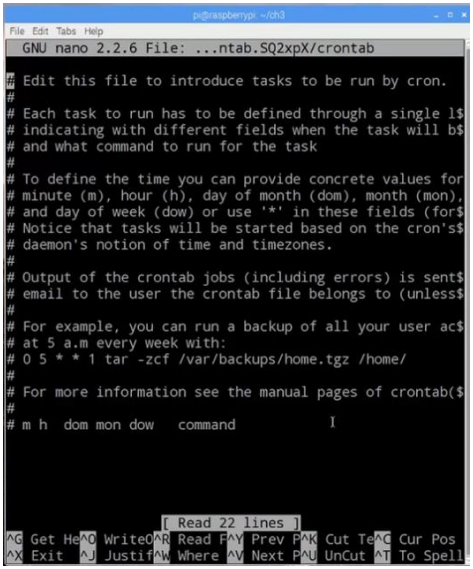
Scheduling an archive Script to run periodically

We schedule programs using what is called the 'Crontab' which is short for Cron table. The Crontab is accessible by typing in crontab, space, and an option '-e' which stands for edit. Then hit the enter key on the keyboard.

```
Crontab -e
```

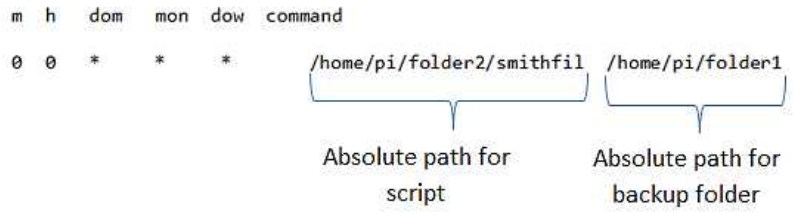


If you haven't run the Crontab before, a prompted might pop up, giving you an option of what editor to use. I prefer to use nano, which is the second option on the list.

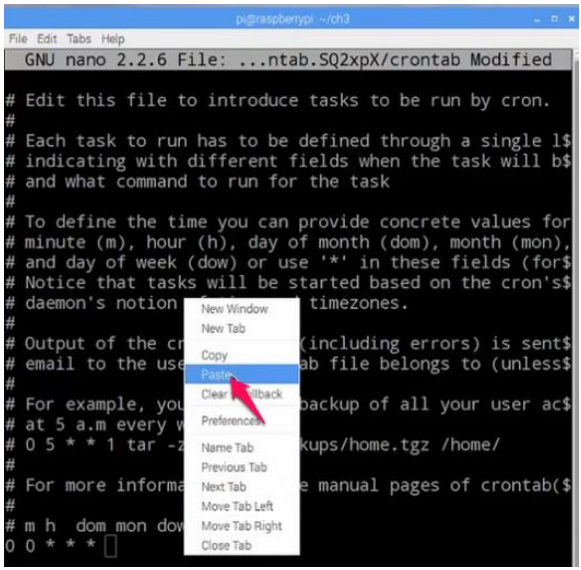


The Crontab has a bit of description and a couple of examples guiding users on the syntax that should be used for running the commands.

Example: Let's schedule a script called "time sensitive archive" to run at midnight every day. To do this, we have to schedule the program using - m (minutes), h (hour), dom (day of the month), mon (month), dow (day of the week) and command as shown in the last line of Crontab editor above.



Because we want the script to run at midnight, we inserted zero minutes, zero hours, and we want to run the script every day, every month and every day of the week. So we use the asterisk (*) symbol to represent every day, every month, and every day of the week. For the command line, copy the absolute file path of the scheduled script and paste it under the command.



After pasting the absolute file path for the command, copy and paste a directory in which to back up the script. So we have scheduled the time-sensitive archive script to run at 0m: 0hrs (midnight), every day of the month, every month and every day of the week. If you want to cue up other schedules, put them on a new line. For instance, if we want to run our sudo APT-GET command to keep our packages up to date, we can do this, let's say every Monday by 3.00am, every month. We write the schedule as follows;

```
m   h   dom   mon   dow   command

0   3   *     *     1     sudo apt.get update && sudo apt.upgrade
```

What the above command line does is to keep our software packages up-to-date by scheduling the period to do so automatically. We schedule a command to run the apt.get

command which gets an updated package list at the 0 min, 3 am, on Monday (Monday is represented by 1 being the first day of the week). The apt.get command does not install any upgrades, what we can do with Cron or any shell script in the terminal is to stack commands together using two epicenters (&&). What this means is run one program then go to the next program and run it. That is, the system will run the 'apt.get update' command before running the 'apt.get upgrade' command. The system will send a prompt asking us if we are sure we want to run an upgrade then we click on 'Y' for yes. But because we trust the makers of this upgrades, we can add the option '- y' for the system to always assume 'Yes' without having to prompt us for our input before running the upgrade.

```
m  h   dom   mon   dow   command
0  3   *     *     1     sudo apt.get update && - y sudo apt.upgrade
```

This means that every Monday at 3' o clock in the morning we will have a list of repositories where we can download packages from, and the packages are then downloaded.
You might also want to reboot your Raspberry Pi periodically. Let's say we want to do that at 10:55 pm on the first of every month. The schedule will be written as follows;

```
m   h    dom    mon    dow   command
55  22   1      *      *     sudo reboot
```

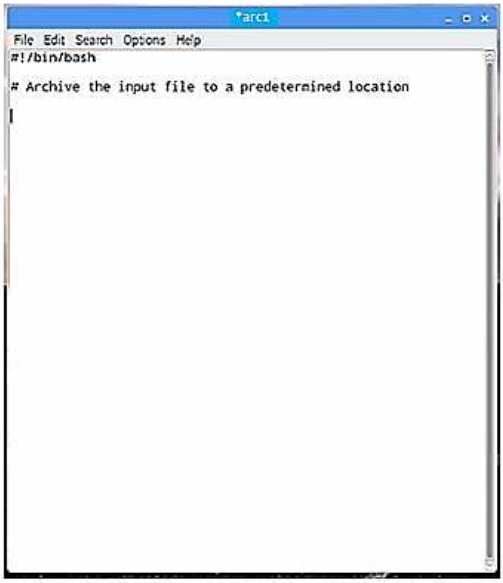
We can create other complicated schedules just by editing the scheduling syntaxes

Writing Useful Backup Scripts

What we will do here is to write what I will call an 'archiving script,' which will take a single argument, which is the file path for a file or directory (location), and then copy whatever is in the directory or location to a predetermined directory or location. We will create a new file called 'arc1' (achieve1) in the 'folder1' directory. Open the arch1 by double-clicking on it and write the following command line;

```
#!/bin/bash
```

What this command line does is to archive the input file to a predetermined location.



This command will make use of a lot of things we have already seen such as arguments and variables. We can set up a variable where we want to maintain this archive by creating a variable called 'archive.' We can set this variable to a file path by entering a file path. For instance

```
Archieve=/home/pi/archieve
```

This means in our home directory we will have an achieve directory. Now, we have gotten the archive location, next is to make the destination if it doesn't exist using the 'make directory' command line to access the variable archive.

```
Mkdir $archive
```

We can use an option for make directory called '-p.' What this does is that if we specify a very long file path with lots of folders that do not exist yet, the -p will create parent directories along the file path until it gets to the last one. It will also efficiently handle the file path if the directories already exist.

```
Mkdir –p $archive
```

Now we can copy the input file, and this uses a command called cp which stands for 'copy.' Remember we are taking in an argument which is a location.

```
cp $1 $archive
```

What the above copy command syntax means is that we will copy the first argument and we will put it in the archive. So it

copies 'from' and 'to.' So we are copying from whatever we input and moving it to 'archive. Note that the archive we are talking about is the variable archive we created. There is another option we need to add to the command line, which is '- r' which stands for recursive. Cp is reserved for copying files, but if we put in a '- r' option, this recursive copy looks inside subdirectories and copies their content before copying the parent directories.

```
cp –r $1 $archive
```

The above command line is fine as it is, but we can include a safety mechanism which checks if an argument was entered or not. We can run our archive script 'archi1' without an input and it will still run and probably bring up some errors along the line. You can prevent this by adding a safety mechanism that would ensure at least one argument is parsed into the script before it runs. It alerts the user if an argument is not parsed. We do this by writing the command line

```
If [$# -ne 1]
```

The above command lines mean 'if we can access a number of arguments ($#) and the argument(s) is not equal to one (-ne 1). We will now add a then statement that will exit the program and alert the user of the absence of argument to parse. This means there is an error. So we can then say;

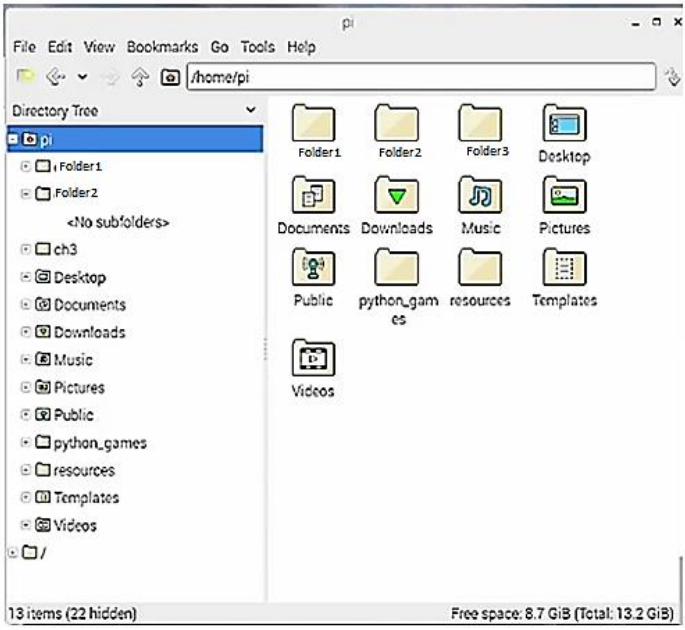
```
Then
Echo  “usage: $0 FILE”
Exit 1
```

When the 'if statement above is true, that is, it doesn't receive the correct number of argument, then the 'then statement' will be executed which will exit the program and print out 1 to indicate an error.
Now let's write the entire code we have been discussing in bits.

```
#!/bin/bash

# Archive the input file to a predetermined location

# Check exactly 1 argument

If   [ $#  -ne  1 ]

then

      echo  "Usage: $0 File"

      exit 1

# The archive location

Archive=/home/pi/archives

# Make the archive (if it doesn't exist)

mkdir –p $archive

# copy the input file

cp –r $1 $archive

-------------------------------------------------
```

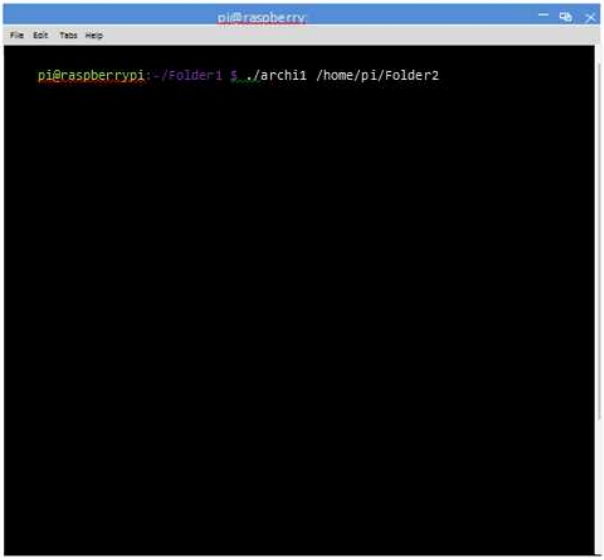
The shell script is ready to run, save it with ctrl + S. Open the terminal and run the archi1 script and you will see the

archive directory is created, but we have to tell the script what to backup. Let us assume that the 'Folder2' directory contains what we intend to move into the archive to back them up using the archi script.

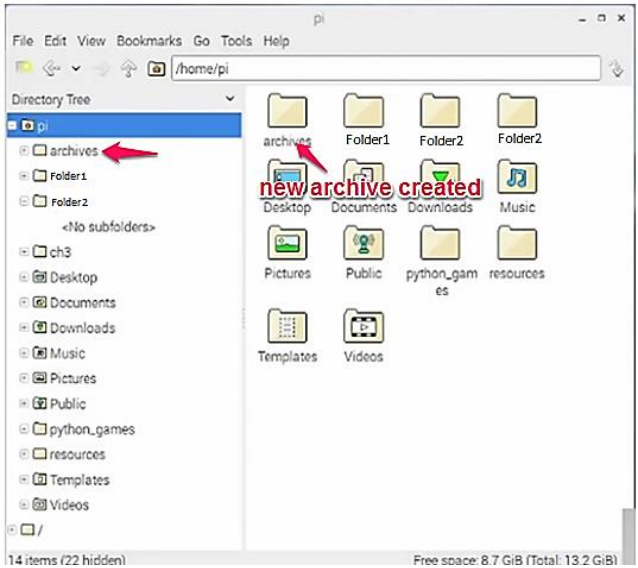


To run the shell script, open the terminal and type in the command

```
pi@raspberrypi:-/Folder1 $ ./archi1 /home/pi/Folder2
```



When you hit the enter key, the command creates a new folder called 'archives' and moves the content of folder2 into the 'archives' folder;



We can modify the arch1 script by adding a feature which creates new directories with the current date on them when it is creating copies. This way, rather than have all the copies thrown into one big directory; they will be filled with lots of subdirectories titled with their creation dates. We can do this with the date command.

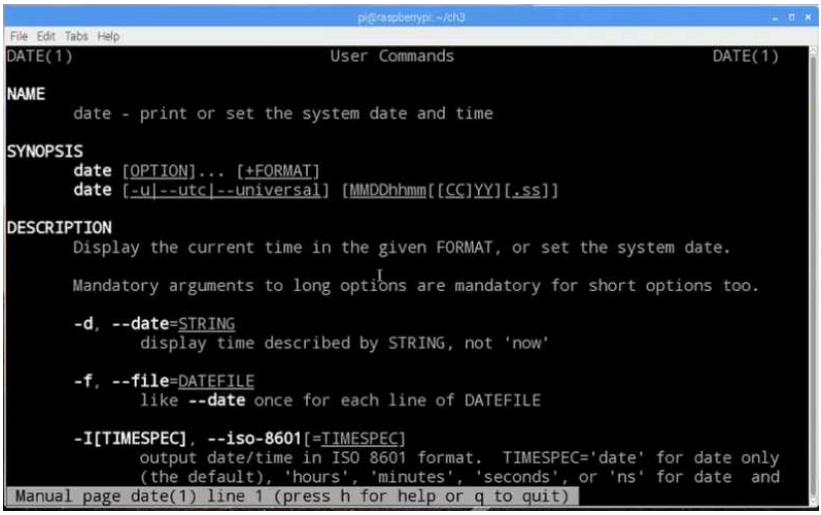
```
pi@raspberrypi:-$ date
```

This will display the date, time and the time zone of the location of the Raspberry Pi

```
pi@raspberrypi:-/folder1 $ date
Thursday 15 March 13:12:18 AEDT 2019
pi@raspberrypi:-/folder1 $
```

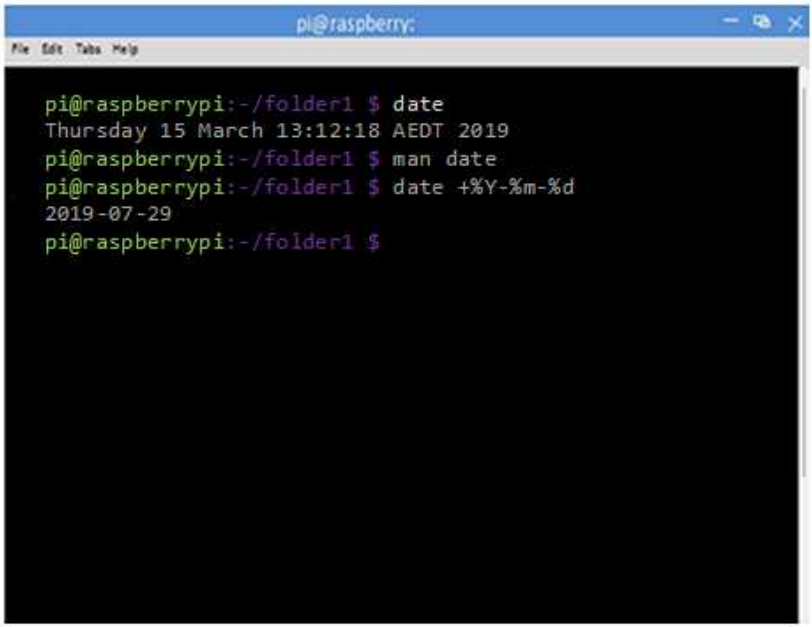
What we will do here is to look at the manual for dates, see how it works and gets it to return exactly the information that we want, that is, the year, the month and the day formatted nicely, so that it grabs that information and create a directory with it. To access the manual command for a date, we type in;

```
pi@raspberrypi:-/folder1 $ man date
```

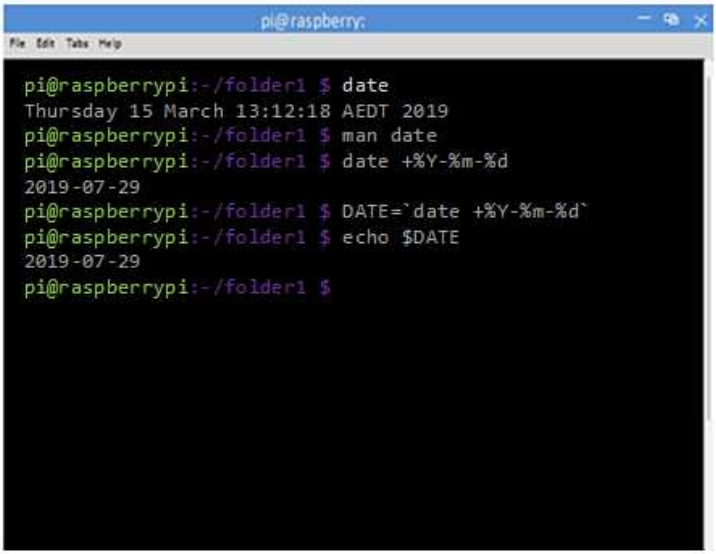


Manual for date

At the top of the manual, we have the synopsis which shows us the way to use the command. We have date followed by [OPTION] and [+FORMAT] which are in square brackets. The [OPTION] that are usable are displayed by -d and – r while the [+FORMAT] gives us hints to format the output, that is, it controls the output. Format is preceded by a plus sign (+). We can control the returning of the output of date by entering some codes into the [+FORMAT] section. These codes begin with a percentage (%) symbol. When you scroll down to the Format section of the manual for date, you will find a list of output formats. For instance, if we want our date to be output in the format Year, Month, Day we will write it thus %Y %m %d.



With this we can create a variable called DATE, rather than setting it equal to the date command, we can declare DATE as a variable and place the date command in between two backticks (`). When we want to run a piece of text as a command and give it to output something, we wrap it in backticks. The backtick character is next to one key on the keyboard; it shares the key with the tilde. We say DATE=`date +%Y-%m-%d`. We can do this in the shell as well as the script because the script is a shell command. This is how we can pass the output of the command 'date' into a variable we call 'DATE.' If we run this and echo $DATE, it will output the current date in the format we have specified.



So DATE=`date +%Y-%m-%d`.is the command we will be using in our script. We can now open up and modify the archi1 script on the nano. Let's rename it archi2. We will be modifying where we set the archive location. We will insert the variable DATE and also append it to the directory Archive=/home/pi/archives. This then changes from

```
# The archive location
Archive=/home/pi/archives
```

To

```
# The archive location
DATE=`date +%Y-%m-%d`
Archive=/home/pi/archives/$DATE
```

We will also modify the copy section form

```
# copy the input file
cp –r $1 $archive
```

to

```
# copy the input file
rsync –avr $1 $archive
```

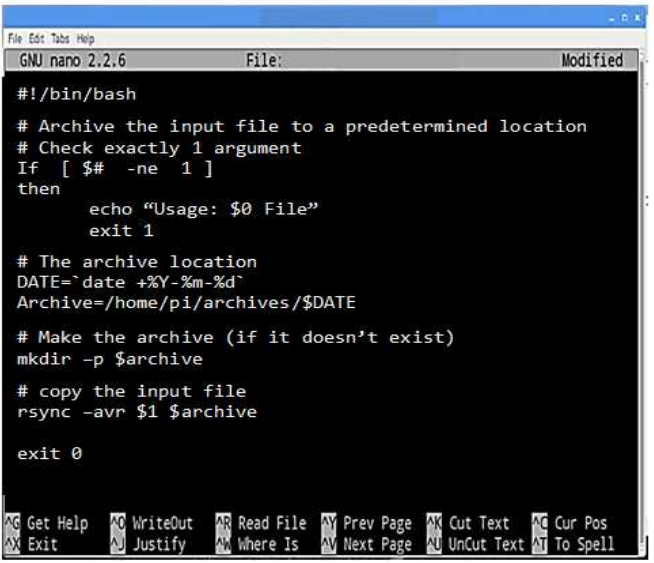
The copy command is a bit clunky; it literally takes an entire copy of everything and creates the copy wherever you tell it to create. We will change the copy command (cp) to a more elegant command called 'rsync'. Rsync is like the copy command, but it is more of a synchronizing command. It checks to see if it actually needs to copy a file before it does it. If you run the script twice in a day and the files haven't changed, it won't need to copy them. The rsync command can also copy over a network so you can use its copy to a remote directory which is recommended for creating archives. What the – avr in the `rsync –avr $1 $archive` line affects how the script will run.

What 'a' does is to keep the permissions for who can execute what and 'v' returns lots of information to tell you what it is doing.

```
#!/bin/bash

# Archive the input file to a predetermined location

# Check exactly 1 argument

If  [ $#  -ne  1 ]

then

      echo “Usage: $0 File”
```

```
      exit 1

# The archive location
DATE=`date +%Y-%m-%d`
Archive=/home/pi/archives/$DATE

# Make the archive (if it doesn't exist)
mkdir -p $archive

# copy the input file
rsync -avr $1 $archive

exit 0
```

The shell script is ready to run, save it with ctrl + S.



From the shell run ./archi2 and pass to it /home/pi/folder2

```
pi@raspberrypi:-/folder1 $ ./archi2 /home/pi/folder2
```

When we run this command, it sends the contents of folder2 directory to the archive it created. When we go into the archive, you will find a directory with the current day's date. And within this directory, you would find folder2 and all the files in it. If we run the command again, no file would be sent to the archive. Because it recognized that all the files already in the archive directory are identical and there would be no need to create a copy because there were no modifications made to the existing file. This is the benefit of using rsync instead of copy.

Variables and Decision Making

In this section, we will work with some variables and run a simple loop. This is has similar logic to programming in Python although with a different syntax. We will start by going to the Raspbian desktop – will use the folder we created for our previous shell scripts called folder1. Click on the folder1 directory or any other folder you have created on your Raspberry Pi desktop – right-click within the folder and create a new empty file. I will name my file 'dec1,' and save. Let's start with the script;

We will declare variables a, b and c. 'a' is assigned 6 and b is assigned 4. A third variable `c=(($a*$b))` will be the multiplication of the content of a and b. To access a variable, we make use of the dollar symbol ($), and we will wrap the entire variables to be accessed in two brackets with another dollar symbol at the front. Alternatively, we can get rid of the

dollar symbols and brackets using the 'Let Statement. (e.g. Let c=a*b. Now we can do some logic with our variables, let's start with an 'If Statement.' An if statement looks like this;

```
If  [       ]
   Then statement
   fi
```

The condition for the 'if statement' is placed in between the angular bracket and we terminate it with a fi. The fi command indicates the end of the 'if statement.' We can write an 'if statement' to check if a is greater than b (a>b). We say;

```
If [ $a  – gt  $b ]
then
   echo a is greater than b
else
   echo a is less than b
fi
```

Where -gt means greater than. Note the spaces between the opening bracket and $a, $a and -gt, -gt and $b, $b and closing bracket. These scripts are syntax sensitive, and you should take every precaution when writing them. We can also write a while statement for this script. For instance;

```
Let c=1
while [ $c –le 10 ]: do
  echo Number $c
let c=c+1 or c++
done
```

Where `c=1` is the initial value of c, `–le` means less than or equal to and `c=c+1 or c++` increments the variable c. The

above 'while statement' means while $c is less than or equal to 10 output the value of c. we end our loop with ''done.'

```
#!/bin/bash
# decision making script
a=6
b=4
c=$(($a*$b)) or Let c=a*b
If [ $a  – gt  $b ]
      then
         echo a is greater than b
      else
      Let c=1
         echo a is less than b
      fi
# loop 9 times
while [ $c –le 9 ]: do
      echo Number $c
      let c=c+1 or c++
```

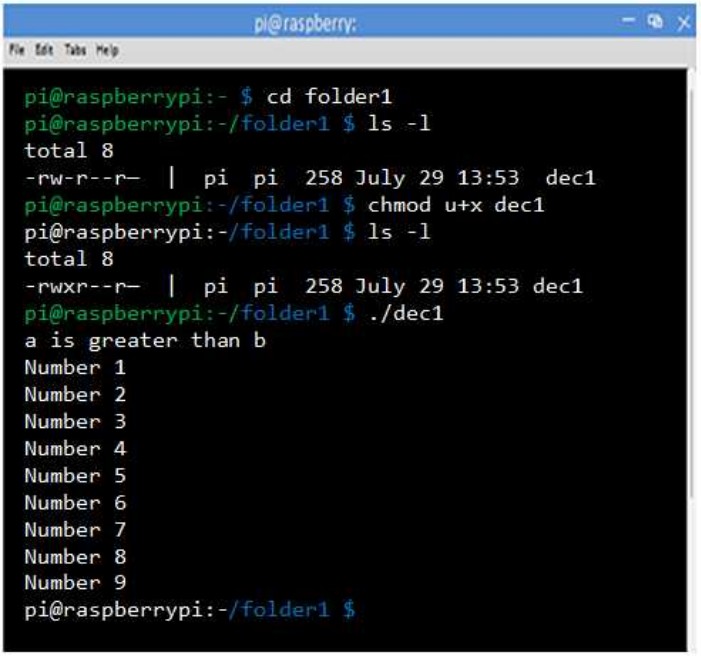
Save the script and open the terminal and run the list command;

```
pi@raspberrypi:-/folder1 $ ls -l
total 8
-rw-r--r– |  pi  pi  258 July 29 13:53  'dec1
```

We can run the **'chmod command'** to make the file **'dec1'** executable, and when we run dec1 as an executable file, we will have an output 'a is greater than b' and 'Numbers 1 to number 9'.

```
pi@raspberrypi:-/folder1 $ ls -l
total 8
-rw-r--r– |  pi  pi  258 July 29 13:53  dec1
pi@raspberrypi:-/folder1 $ chmod u+x dec1
```

```
pi@raspberrypi:-/folder1 $ ls -l
total 8
-rwxr--r—  |  pi  pi  258 July 29 13:53 dec1
pi@raspberrypi:-/folder1 $ ./dec1
a is greater than b
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
pi@raspberrypi:-/folder1 $
```



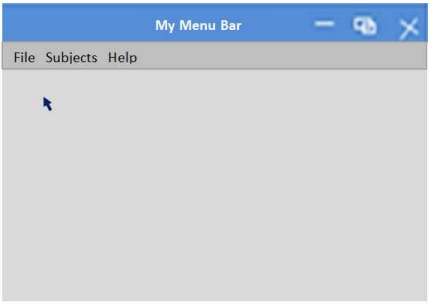
Chapter Six

Introduction to TKInter

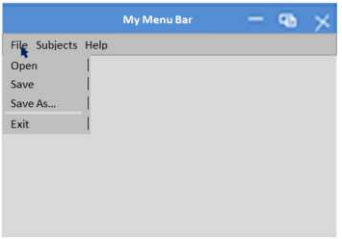
What is TKInter? This is the standard package for programming Graphical User Interfaces on Raspberry Pi. In this chapter, we will cover programming simple graphical user interface with familiar features like buttons, radio buttons, checkboxes, and sliders, which are all referred to as widgets in TKInter. When we create a widget, we have the option of attaching to it certain Python functions we can run. We refer these functions to as 'EVENTS. For instance, if we are programming a button, pressing the button is an event, and any time that event is triggered, we can choose to execute a piece of python code. For instance, if we have a written python function that, anytime we press that button, the function will be executed.

TKInter Menu Bars

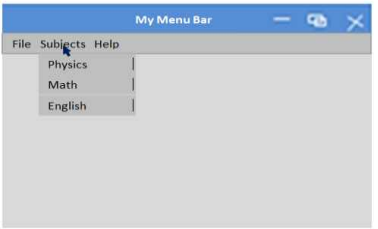
Menu bars are common in Graphic User Interface environments. They are bars that conveniently arrange various options into Menus and submenus. We will be demonstrating a simple Menu Bar which would hold menu items such as File, Subjects, and Help.



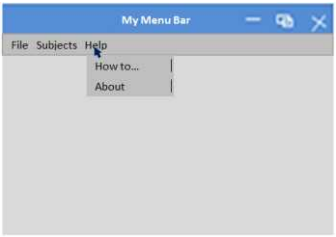
These menu items would hold sub-menu items. For instance, the 'File' would hold sub-menus such as 'Open,' 'Save,' 'Save As' and 'Exit.'



Note we use the code `filemenu.add_separator()` to add a horizontal separator which is a visual feature. An example is a horizontal separator between 'Save As...' and 'Exit.' Horizontal separators are placed at any part of the sub-menu.



The Subjects menu will hold submenus 'Physics', 'Math' and 'English.'



Finally, in our demonstration, we have the 'Help' Menu with sub menus 'How to...' and 'About.'

Now let's look at how to create the same thing. Remember, you can play around with the codes to add menus and submenus of your choice.

```
--------------------------------------------------------------------------------
from tkinter import *

win = Tk()
Win.title("My Menu Bar")

def placeholder():
    popup = Toplevel()
    popup.title("placeholder")
    #Fill the popup window with the Canvas() example image
    C = Canvas(popup, bg="blue", height=250, width=300)
    coord = 10, 50, 240, 210
    arc = C.create_arc(coord,start=0, extent=150, fill="red")
    C.pack()
    Popup.mainloop()

menubar = Menu(win)

# Create menu entry and sub-menu options
# Create the File menu and submenus
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="open" , command=placeholder)
filemenu.add_command(label="Save" , command=placeholder)
filemenu.add_command(label="Save As…" , command=placeholder)
filemenu.add_separator()
filemenu.add_command(label="Exit" , command=win.quit)
menubar.add_cascade(label="File" , menu=filemenu)
```

```
# Create the Subject Menu and submenus
subjectmenu = Menu(menubar, tearoff=0)
subjectmenu.add_command(label="Physics" , command=placeholder)
subjectmenu.add_command(label="Math" , command=placeholder)
subjectmenu.add_command(label="English" , command=placeholder)
menubar.add_cascade(label="Subject" , menu=subjectmenu)

# Create the Help Menu and submenus
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="How to…" , command=placeholder)
helpmenu.add_separator()
helpmenu.add_command(label="About" , command=placeholder)
menubar.add_cascade(label="Help" , menu=helpmenu)

# Display the menu
Win.config(menu=menubar)
```

We can now look at what each code line is doing.

`from tkinter import *` is the standard import for TKInter and

`win = Tk()`

`Win.title("My Menu Bar")` is the standard Graphical User Interface definition for the Menu bar. We have the little window that pops up whenever we click on any of the menu items.

```
def placeholder():
    popup = Toplevel()
    popup.title("placeholder")
    #Fill the popup window with the Canvas() example image
    C = Canvas(popup, bg="blue", height=250, width=300)
    coord = 10, 50, 240, 210
    arc = C.create_arc(coord,start=0, extent=150, fill="red")
    C.pack()
    Popup.mainloop()
```

We can code the placeholder to perform several functions. The placeholder can also hold several other codes which can be called when each menu item is clicked. We can also duplicate the placeholder to perform different functions. When we click on a submenu object, a new window appears which

performs the function within the placeholder. The new window is created by the Top-level function, as shown in the function `popup = Toplevel()`. You can give the Toplevel a title, create and pack widgets within it. Within our placeholder function Toplevel, we created a Canvas widget which pops up. We also set the background to blue and set the height and width to 250 and 300 respectively.

```
C = Canvas(popup, bg="blue", height=250, width=300)
```

In this demonstration, we are creating a menu bar as a menu object `menubar = Menu(win)` and it automatically puts the menu at the top of the window. Then we fill the bar with menu options. What we do here is create the menu object which itself a menu object.

```
xxxmenu = Menu(menubar, tearoff=0)
```

Rather than loading the `menu object` into `win` like we did for `menubar`, we are loading it into `menubar` itself, which means a Menu widgets can contain other menu widgets. We also have a function called `tearoff=0`, which allows us to pull the menu away if we so wish. Next, we run the add command function `xxxmenu.add_command`. This allows us to create the options within that same menu (e.g., for the File Menu we can create submenus like Open, Save, Save as... and Exit). We give a label to each of the add command options, and we can write some codes which will replace the place holder.

```
filemenu.add_command(label="open" , command=placeholder)
```

From our code, you could see that all the commands are set to a placeholder. Which means every option for each menu will run the same placeholder function as defined by the function below;

```
def placeholder():
    pop up = Toplevel()
    popup.title("placeholder")
    #Fill the popup window with the Canvas() example image
    C = Canvas(popup, bg="blue", height=250, width=300)
    coord = 10, 50, 240, 210
    arc = C.create_arc(coord,start=0, extent=150, fill="red")
    C.pack()
    Popup.mainloop()
```

We can duplicate the placeholder into different functions with some more useful code written, which you can pass into the different menu and submenu objects.

Each menu item is added to the menu bar using the add_ cascade command.

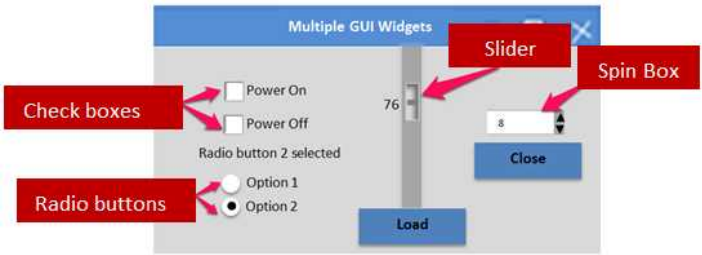
```
Menubar.add_cascade(label="xxx" , menu=xxxmenu)
```

The order with which the add_cascade command is run decides how the menu item is positioned on the menu bar.

Codes for Different TKInter Widgets

In this section, we will write a script that displays different TKInter widgets in a Graphical User Interface. We will write codes for checkboxes, radio buttons, a slider, spin box, and

text. The use of the radio button is for selecting mutually exclusive options, the slider which can go between zero and a hundred is used for adjusting continuous parameters; the use of checkboxes are for controlling things individually, the spinbox is for selecting a range of predefined values. Finally, we can also display texts on the interface dynamic to display status using the label widget. For instance, if we select the radio button one, the text changes to Radio button 1 selected, and when we select button two, the text changes to Radio button 2 selected.



Let's look at how each of the above widgets is built in the script below;

```
-------------------------------------------------------------
## A demonstration of multiple Tkinter widgets ##

from tkinter import *
import tkinter.font

## GUI Definitions ##
win = Tk()
win.title("Multiple GUI Widgets")
myFont = tkinter.font.Font(family = 'Helvetica', size = 12,
weight = "bold")

leftFrame = Frame(win)
leftFrame.pack(side = LEFT)
rightFrame = Frame(win)
rightFrame.pack(side = RIGHT)
```

```
midFrame = Frame(win)
midFrame.pack(side = RIGHT)

## Definitions for widgets ##
checkBox1 = Intvar()
checkBox2 = IntVar()
sliderval = DoubleVar()
rad = IntVar()

## Event Functions ##
def buttonPress():
    print('The Slider Value is {}' .format(sliderval.get()))

def checkToggle():
    print ('\nCheckbox #1 #2\n   {}  {}'
.format(checkBox1.get(), checkBox2.get()))

def checkRadio():
    select = "Radio button  " + str(rad.get()) + " selected"
    label.config(text = select)

def close():
    win.destroy()   #close the Graphical User Interface

## Attaching Variables to WIDGETS ##
# triggering the connected commands with a press on the button
myButton = Button(midFrame, text='Load' , font=myFont,
command=buttonPress, bg='bisque2' , height=1)
myButton.pack(side = BOTTOM)

#The Two checkboxes would call command when toggled
check1 = Checkbutton(leftFrame, text='Power On  ',
variable=checkBox1, command=checkToggle)
check1.pack()
check2 = checkbutton(leftFrame, text='Power Off' ,
variable=checkBox2, commandToggle)
check2.pack()

# Creating the Radio Buttons for mutually exclusive options
Label = Label(leftFrame)
Label.pack()
Label.comfig(text="No option is selected")
R1 = Radiobutton(leftFrame, text="Option 1", variable=rad,
value=1, command = checkRadio)
R1.pack()
R2 = Radiobutton(leftFrame, text="Option 2",variable=rad,
value=2, command = checkRadio)
```

```
R2.pack()

#Creating Slider bar for adjusting parameters
Slider = Scale(midFrame, variable=Sliderval,  from_100.0,
t0=0.0)
Slider.pack()

#Creating spinbox for selecting range of predetermined values
SpinNum = Spinbox(rightFrame,  from_=1, to = 10, width=6)
spinNum.pack(side = TOP )

# creating close button to exit the Graphical User Interface
closeButton = Button(rightFrame, text='Close', font=myFont,
command=close, bg='blue', height=1, width=6)
closeButton.pack(side = BOTTOM )

win.mainloop() # Continuous Loop
```

Save and run the script. We can now look at what each code line is doing.

```
win = Tk()
win.title("Multiple GUI Widgets")
myFont = tkinter.font.Font(family =  'Helvetica',  size = 12,
weight = "bold")
The above code is used to build the entire window frame
holding the different widgets.

checkBox1 = Intvar()
checkBox2 = IntVar()
sliderval = DoubleVar()
rad = IntVar()
```

The above are the variables that are updated whenever we interact with the widgets and TKInter needs the intVar and DoubleVar to hold the values.

A frame is a widget that can contain other widgets and frames. Unlike grids, using frames is a better way to organize our widgets easily. While we use the grid to place widgets explicitly, we can fill a frame with widgets and move the frame containing that widget(s) easily. In our GUI, we have

arranged things in what is essentially three columns (or three frames) which we created within the window using the function below;

```
leftFrame = Frame(win)
leftFrame.pack(side = LEFT)
rightFrame = Frame(win)
rightFrame.pack(side = RIGHT)
midFrame = Frame(win)
midFrame.pack(side = RIGHT)
```

We have the left frame containing the checkboxes, the dynamic text label, and the radio buttons. We have at the center of the window a middle frame which has the slider and its 'Load' button, while the frame on the right contains the spin box and the exit button. So when you look at each widget function, you will find the frames in which they are placed (Left, mid and right frame) and we do this with a command called 'pack. In other words, you can make a frame and pack it. You can also make widgets and pack them into frames. This is an easier way to group items with similar functionality or intended for some job into the same frame. We can also place the widget at different sides within the frame either TOP or BOTTOM of the frame.

Just like we can attach commands to widgets, we can also attach variables to certain widgets as well. This is seen in the code in the widget definition section as seen below;

```
myButton = Button(midFrame, text='Load' , font=myFont,
command=buttonPress, bg='bisque2' , height=1)
myButton.pack(side = BOTTOM)
```

We have two check boxes built with the code below;

```
check1 = Checkbutton(leftFrame, text='Power On  ',
variable=checkBox1, command=checkToggle)
check1.pack()
check2 = checkbutton(leftFrame, text='Power Off' ,
variable=checkBox2, commandToggle)
check2.pack()
```

We also have two radio buttons built with the code below;

```
Label = Label(leftFrame)
Label.pack()
Label.comfig(text="No option is selected")
R1 = Radiobutton(leftFrame, text="Option 1", variable=rad,
value=1, command = checkRadio)
R1.pack()
R2 = Radiobutton(leftFrame, text="Option 2",variable=rad,
value=2, command = checkRadio)
R2.pack()
```

We've also got that slider bar with an upper and lower limit of hundred and zero, respectively. The slider is called a 'scale' in TKInter. The slider is packed into the mid-frame.

```
Slider = Scale(midFrame, variable=Sliderval, from_100.0,
t0=0.0)
Slider.pack()
```

Similarly, with the spin box we are creating an object called spinNum which we are packing into the right frame with and upper and lower limit of ten and zero respectively. We also set some width to deal with large numbers.

```
spinNum = Spinbox(rightFrame, from_=1, to = 10, width=6)
spinNum.pack(side = TOP )
```

And of course the exit button and its parameters;

```
closeButton = Button(rightFrame, text='Close', font=myFont,
command=close, bg='blue', height=1, width=6)
closeButton.pack(side = BOTTOM )
```

The event functions hold functions that would be performed when each of the widgets is acted upon.

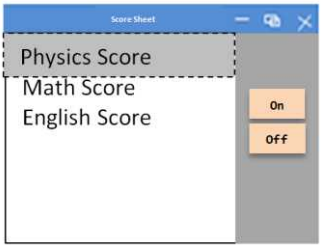
```
def buttonPress():
    print('The Slider Value is {}'  .format(sliderval.get()))

def checkToggle():
    print  ('\nCheckbox #1 #2\n   {}  {}' .format(checkBox1.
    get(), checkBox2.get()))

def checkRadio():
    select = "Radio button  " + str(rad.get()) + " selected"
    label.config(text = select)

def close():
    win.destroy()   #close the Graphical User Interface
```

TKInter Events and Bindings

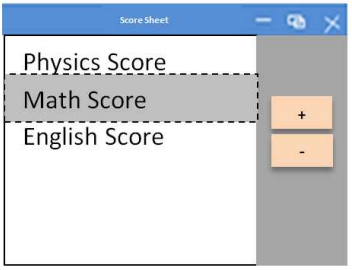
Events are just only when we are clicking on interfaces, but events can come from the user directly 'via the keyboard and mouse. For instance, a double-click event or a certain key is pressed. These events can be bound to certain functions, just like a widget can be bound to a function.

Example:

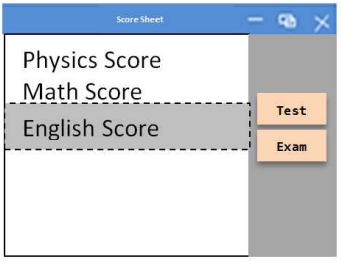
Let's write an event for a list box which has various items we can select. But if we double-click on an option, we get the buttons by the right to change. When we double-click on physics Score, we have the "On" and "Off" Button displayed



When we double-click on Math Score, the button changes to plus (+) and minus (-).



When we double-click on "English Score" the button changes to "Test" and "Exam."



We can now write a python program code to perform these functions.

```
## Demonstrates a small list-style menu with mouse-click
events
from tkinter import *
import tkinter.font
```

```
### GUI DEFINITIONS ###
win = Tk()
win.title("score sheet")
myFont = tkinter.font.font(family='Helvetica',  size = 12,
weight = "bold")

menuFrame = Frame(win)
menuFrame.pack(side = LEFT)
phyFrame = Frame(win)
mathFrame = Frame(win)
engFrame = Frame(win)

###EVENT FUNCTIONS ###
def buttonPress(): #Placeholder button event
    print("Button Pressed")

def callback(event):
print("clicked at", event.x,  event.y)

def menuManage(dummy):  # Manage what options are visible
    # Start by hiding all buttons
    sel = Lb.curselection()
    phyFrame.pack_forget()
    mathFrame.pack_forget()
    engFrame.pack_forget()

    # Show the relevant buttons depending on list selection
    if sel[0]  == 0:
       phyFrame.pack(side=RIGHT)
    elif sel[0] == 1:
        mathFrame.pack(sode=RIGHT)
    elif sel[0]  == 2
        engFrame.pack(side=RIGHT)

### WIDGETS ###
# Listbox for text-item selection
Lb = Listbox(menuFrame)
Lb.insert(1,"Physics Score")
Lb.insert(2,"Math Score")
Lb.insert(3,"English Score")
Lb.bind("<Double-Button-1>", menuManage) #Attach double-click
event
##Lb.bind("Double-Button-1>" ,  callback)
Lb.pack()

#Placeholder buttons for functionality
```

```
phyOn = Button(phyFrame, text='On' , font=myfont,
command=buttonPress, bg='bisque2', height=1)
phyOff = Button(phyFrame, text='Off', font=myFont,
command=buttonPress, bg='bisque2', height=1)
phyOn.pack(side = TOP)
phyOff.pack(side = BOTTOM)

mathPlus = Button(mathFrame, text='+' , font=myFont,
command=buttonPress, bg='bisque2', height=1)
mathMinus = Button(mathFrame, text ='-' , font=myFont,
command=buttonPress, bg='bisque2' , height=1)
mathPlus.pack(side = Top)
mathMinus.pack(side = Bottom)

engOn = Button(engFrame, text='Test', font=myFont,
command=buttonPress, bg='bisque2',height=1)
engOff = Button(engFrame, text='Exam', font=myFont,
command=buttonPress, bg='bisque2', height=1)
engOn.pack(side = TOP)
engOff.pack(side = BOTTOM)

win.mainloop()
```

In the menuManage event, we start by hiding all the buttons.

```
def menuManage(dummy): # Manage what options are visible
    # Start by hiding all buttons
    sel = Lb.curselection()
    phyFrame.pack_forget()
    mathFrame.pack_forget()
    engFrame.pack_forget()
```

Then we pack a frame to make it appear, showing two buttons at a time. The pack.forget makes the button disappear. We also created a variable called 'sel' and we gave it the value of the selection of the List Box

`Lb.curselection()`

So if we double-click on Math Score, which is the second item which is essentially the 1st index, sel will fetch integer

value 1 and every time we double-click on the List Box, the variable will be updated with the index of whichever item we double-clicked on.

```
# Show the relevant buttons depending on list
selection
 if sel[0] == 0:
    phyFrame.pack(side=RIGHT)
 elif sel[0] == 1:
     mathFrame.pack(sode=RIGHT)
 elif sel[0] == 2
     engFrame.pack(side=RIGHT)
```

After the menuManage event, we carry out some if-else chain. We do this chain to show the relevant button, for instance, if we clicked on the first or 0th item, it would show the Physic Score frame, which is On/Off. If we had selected the second which is 1st, then sel will return a one (1) in its integer. We use a square bracket for sel because it is not just some integer, but an object containing a bit of other information. So the 0th entry in sel is the index of whichever selection was initially made.

In the widget section, we created a variable called LB (List Box), and we insert into the List Box various options.

```
Lb.insert(1,"Physics Score")
Lb.insert(2,"Math Score")
Lb.insert(3,"English Score")
```

We have physics score, math score, and English Score. Of course, these are variables—it could be any menu item. Another thing we are doing with LB is the bind function

```
Lb.bind("<Double-Button-1>", menuManage)
#Attach double-click event
##Lb.bind("Double-Button-1>" , callback)
```

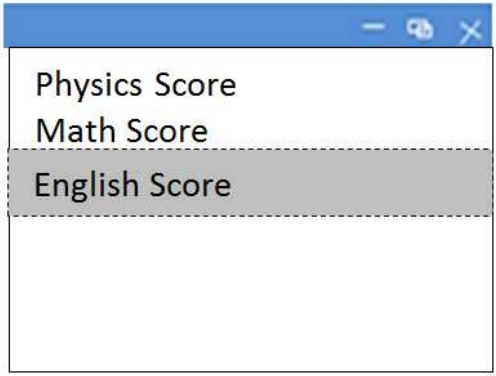
What this does is to bind an event to a function. For instance, we are binding Double-Button-1event, which is a double left-click to menuManage. What we are also doing is allowing that binding only to occur while we are on the list box (LB). This means if we double click outside the list box, nothing happens; we only get a response when we click on the list box options.

```
def callback(event):
print("clicked at, event.x, event.y)
```

The callback is triggered by double button 1. In other words, double button 1 is the event passed to the callback. We can access the properties of that event like the x and y coordinates, and they are printed straight to the shell. This means the coordinates of where the mouse was clicked can be output on the shell using the callback event when we run the script and click on a different part of the widget.

```
Python 3.4.2  (default, July 19 2019. 13:31:11
(GCC 4.9.1) on linux
Type  "copyright", "credits" or "license()" for more
information
>>>>==================RESTART====================
>>>>
```

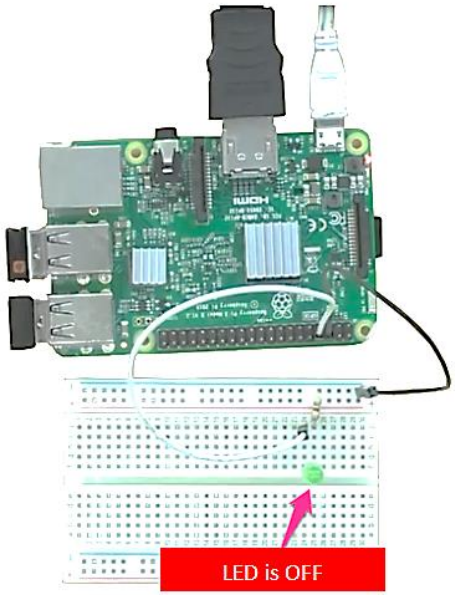
```
clicked at 25 82
clicked at 126 74
clicked at 4 6
clicked at 61 96
```

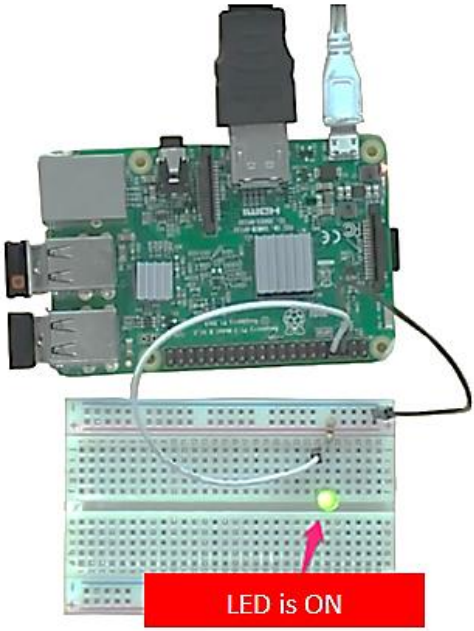


GUI with TKInter and Python

In this section, we will be looking at how to use TKInter and a Python script to control an LED.

Example1: We will create a very simple interface with a button that toggles ON an LED and another button to turns off the LED and also close the application. For this demonstration, we will be using a breadboard, and a single LED connected to GPIO 14 on the Raspberry Pi. When we click the 'Turn LED on' button, the LED on the breadboard is turned ON. When we click on the 'Turn LED Off,' the LED is turned off, and when we click on the 'Exit' button, the LED on the Breadboard is turned OFF, and GUI closes.


LED is OFF


LED is ON

Below is the Python script to achieve the above function

```
from tkinter import *
import tkinter.font
from gpiozero import LED
import RPi.GPIO
RPi.GPIO.setmode(RPi.GPIO.BCM)

## Hardware definition
led = Led(14)

## GUI DEFINITIONS ##
Win = Tk()
Win.title("LED Toggler")
myFont = tkinter.font.Font(family = 'Helvetica', size = 12,
weight= "bold")

## EVENT FUNCTIONS ###
def ledToggle():
    if led.is_lit:
       led.off()
       ledButton["text"] = "Turn LED on"
    else:
       led.on()
       ledButton["text"] = "Turn LED off"

def close():
    RPi.GPIO.cleanup()
    Win.destroy()

### WIDGETS ###
ledButton = Button(win, text = 'Turn LED On', font = myFont,
command = ledToggle, bg = 'bisque2', height = 1, width = 20)
ledButton.grid(row=0,column=1)

exitButton = Button(win, text = 'Exit', font = myFont, command
= close, bg = 'red', height = 1, width = 6)
exitButton.grid(row=1,column=1)

win.protocol("WM_DELETE_WINDOW", close) #Exit Widget

win.mainloop() #Loop forever and keeps the GUI running
```

Save and run the script. Now let's break down the script line by line for better understanding of what each row is doing.

Line 1:

```
from tkinter import *
```

The above synthax allows us to access all functions from 'tkinter' without having to import individual tkinter functions.

Line 3:

```
from gpiozero import LED
```

gpiozero is similar to Rpi.GPIO except that it comes with functions that are made for specific roles (e.g., LED). In the next line, we imported the Rpi.GPIO which is for the script's clean-up method. Just like Tkinter works, when the Graphical User Interface is closed, we need the Rpi.GPIO for clean up.

Line 1 - 5:

```
from tkinter import *
import tkinter.font
from gpiozero import LED
import RPi.GPIO
RPi.GPIO.setmode(RPi.GPIO.BCM)
```

These will allow us to build a Graphical User Interface, flash a LED and clean up when we close the GUI.

Line 7:

```
led = Led(14)
```

Is where we define our hardware.

Line 9 - 11:

```
Win = Tk()
Win.title("LED Toggler")
myFont = tkinter.font.Font(family = 'Helvetica', size = 12,
weight= "bold")
```

Is where we create the major Graphical User Interface window without the buttons in it. The `Win = Tk()` creates an

object in which we can build our GUI. We give the window a title `Win.title("LED Toggler")`which appears across the top of the window and we also create fonts to pack into our GUI using the `myFont = tkinter.font.Font function`. In the bracket after `myFont = tkinter.font.Font function` is where we specify the font family, font size, and weight.

Line 23 - 26: Is where we create the widgets (buttons) using the code lines

```
ledButton = Button(win, text = 'Turn LED On', font = myFont,
command = ledToggle, bg = 'bisque2',  height = 1,  width = 20)
ledButton.grid(row=0,column=1)

exitButton = Button(win, text = 'Exit', font = myFont, command
= close, bg = 'red', height = 1, width = 6)
exitButton.grid(row=1,column=1)
```

To create a button with TKinter, we invoke the function `Button` after which we will specify where the button should be placed and attach it to some event function.

```
ledButton = Button(win, text = 'Turn LED On', font = myFont,
command = ledToggle, bg = 'bisque2',  height = 1, width = 20)
```

So our button will be placed in `win` which is the main window—we will also give it a text, which will appear on the button (e.g., `Turn LED On`) and we can apply a font to that text (`font = myFont`). Now we need to attach a command to the button so that on pressing the button, it calls this function(s).

Lets call the command the button will activate when pressed `'ledToggle' (command = ledToggle)`. Now we need to define `ledToggle` as an event function which the buttons will call.

```
def ledToggle():
    if led.is_lit:
       led.off()
       ledButton["text"] = "Turn LED on"
    else:
       led.on()
       ledButton["text"] = "Turn LED off"
```

When we create a GUI, we must be explicit about its properties, text color, background color (bg), height, width, etc. Now we are done creating the buttons, but we haven't placed it anywhere on the graphical user interface. There are a couple of ways to do this, but in this demonstration, we used the grid structure. We place buttons on the GUI with the aid of coordinates or rows and columns numbers. So we will place our button using the `ledButton.grid` function. The row will be zero (i.e., the top), and column will be at one.

```
ledButton.grid(row=0,column=1)
```

We do the GPIO clean up by creating an exit button, and this exit button will trigger the closing of the interface and also clean up the GPIO. Here is the code to create the exit widget;

```
exitButton = Button(win, text =  'Exit', font = myFont,
command = close, bg =  'red', height = 1, width = 6)
exitButton.grid(row=1,column=1)
```

then we define the function for the close event;

```
def close():
    RPi.GPIO.cleanup()
    Win.destroy()
```

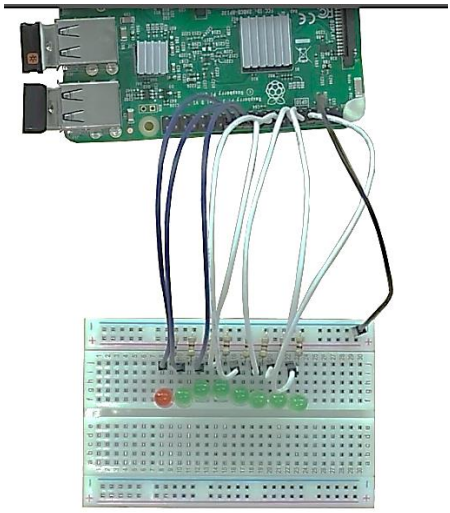
This means we can close the program at any point and if the LED is ON, it will turned Off, and the GPIO pin will reset to its reset state. What the `Win.destroy()` does is clean up the GPIO and also close the window when the exit button is clicked on.

Finally, we would need to capture an event which turns off the LED when the "x" button at the top right corner of the window is used to close the window instead of the Exit button. This event is built into TKInter called **"WM_DELETE_WINDOW" and we attach it to our close button.**

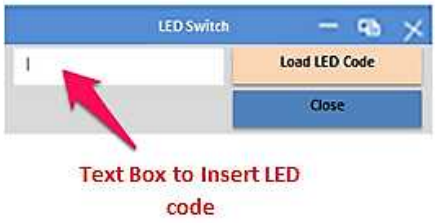
```
win.protocol("WM_DELETE_WINDOW", close)
```

And the function `win.mainloop()` keeps the Graphical User Interface running.

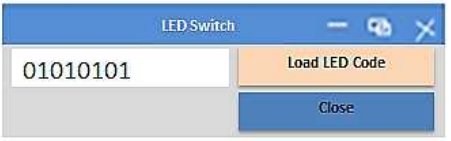
Example 2: In this illustration, we will demonstrate how to set up several LEDs on the breadboard and toggle each of the LEDs On or Off using TKInter and a Python script. For this project, we will be using the Raspberry Pi, breadboard, eight LEDs, eight resistors (4ohms) and wires to connect the LEDs to the GPIO of the Raspberry Pi.

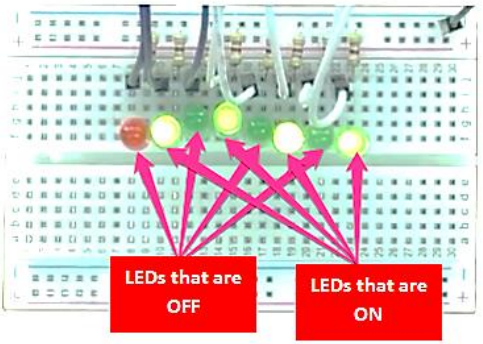


We can turn ON and OFF the LEDs using strings which will be keyed into the text box of a Graphical User Interface called "LED Switch" that we will be creating. The textbox will accommodate only eight ones or zeros with each representing the LED's state (whether ON or OFF) For instance, anywhere there is a one (1), that LED will be ON and anywhere there is a zero, that LED will be OFF.



When we insert the 01010101, what we get is four LEDs lit up alternately as shown below.





These LED codes can be alternated, which will reflect on the LEDs that would be turned ON or OFF. Now, let's write the script for this project.

```
from tkinter import *
import tkinter.font
from gpiozero import LED
import RPi.GPIO
RPi.GPIO.setmode(RPi.GPIO.BCM)

### HARDWARE DEFINITIONS ###
# LED pin definitions
led0 = LED(7)
led1 = LED(8)
led2 = LED(25)
led3 = LED(23)
led4 = LED(24)
led5 = LED(18)
led6 = LED(15)
led7 = LED(14)
# Arrange LEDs into a list
leds = [led7,led6,led5,led4,led3,led2,led1,led0]
```

```
### GUI DEFINITIONS ###
Win=Tk()
Win.title("LED Switch")
myFont=tkinter.font.Font(family = 'Helvetica', size = 12,
weight = "bold")
ledCode = StringVar()

### Event Functions ###
def ledShow():
    ledCode = code.get()
    print("LED code: ", ledCode) #Debug

    i=0 #loop-counter
    # For each character in the ledCode String, check if = 1
    # and if so, turn on the corresponding LED
    for c in ledCode:
        if c ==  "1":
            leds[i].on()
        else:
             leds[i].off()
        i+=1
def close(): # Cleanly close the GUI and cleanup the GPIO
    RPi.GPIO.cleanup()
    Win.destroy()

### WIDGETS ###
ledButton = Button(win, text='Load LED code', font=myFont,
command=ledShow,  bg='bisque2',  height=1)
ledButton.grid(row=0,column=1)

code = Entry(win, font=myFont,  width=10)
code.grid(row=0,column=0)

exitButton = Button(win, text='Exit',font=myFont,
command=close, bg= 'green', height=1, width=6)
exitButton.grid(row=3,column=1,sticky=E)

win.protocol("WM_DELETE_WINDOW",  close) #Cleanup GPIO when
user closes window
win.mainloop() #loos forever
```

Save and run the script. We will break down the script line by line for a better understanding of what each row is doing.

We will skip some of the lines which have been explained in our previous project.

```
Line 7 – 15: We created some LED objects and the pins (GPIO)
each LED is attached to is written in brackets.
led0 = LED(7)
led1 = LED(8)
led2 = LED(25)
led3 = LED(23)
led4 = LED(24)
led5 = LED(18)
led6 = LED(15)
led7 = LED(14)
```

Line 17: We arrange the LED objects into a 'List' with the notation `leds = [led7,led6,led5,led4,led3,led2,led1,led0]. So the list is called 'leds'.`

Line19 – 22: We create the Graphical User Interface Window with the title "LED Switch." In it, we also create an LED code variable called `StringVar()`.It is the variable coming from the text box. We create the text box with a widget called 'entry.' An entry widget is used to hold a string.

```
code = Entry(win, font=myFont, width=10)
code.grid(row=0,column=0)
```

```
The "Load LED code" button is what drives the entire show. So
it is connected to called 'ledShow.' The ledShow event
function is then defined thus;
def ledShow():
    ledCode = code.get()
    print("LED code: ", ledCode) #Debug

    i=0 #loop-counter
    # For each character in the ledCode String, check if = 1
    # and if so, turn on the corresponding LED
    for c in ledCode:
        if c == "1":
            leds[i].on()
        else:
             leds[i].off()
        i+=1
```

Because we have code as the textbox entry object, we can perform a 'get' on that using the `code.get()` function which prints the code ("LED Code") that is entered into the text box.

The decision is taken in the code block below

```
for c in ledCode:
        if c == "1":
            leds[i].on()
        else:
             leds[i].off()
```

This checks whether the character entered is one or not. We first create a loop counter which we set to zero i=0. The Zero is the zeroth element of the list which represents the first entry of that list. So we create an LED counter and then a "for-loop." Because led code is a string, c is going to be a character. That means, for every character, in the string, check if the character is equal to one (1). If it is, turn LED On `leds[i].on()`and this is where the list `leds = [led7,led6, led5,led4,led3,led2,led1,led0]` is powerful. The list determines which of the LED should be on if it meets the condition, and which should be Off when it didn't meet the condition. The list accesses the index of the1st entry. So if the character is not a one, then it turns the corresponding LED Off, else the corresponding LED is turned ON, and then we increment our loop counter `i+=1.` So we have our for loop automatically stepping through the string, and we have our loop counter matching each position and toggling that LED based on the value of that character.

Chapter Seven

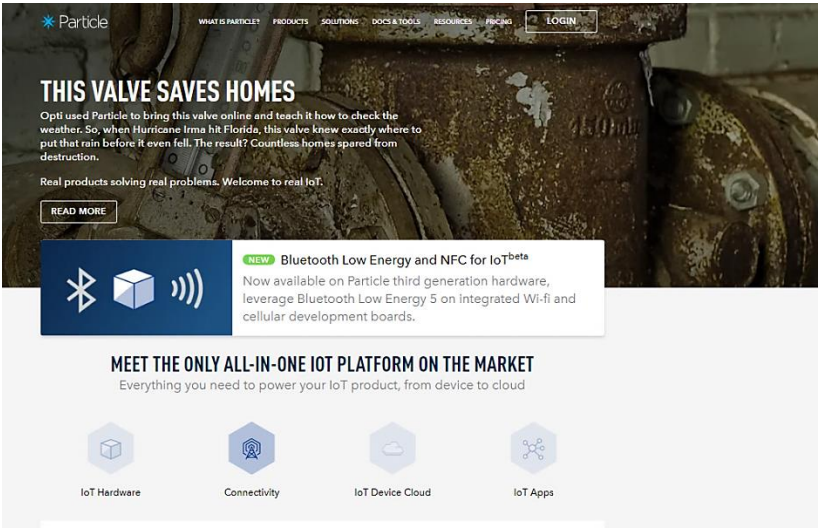
Internet of Things (IoT)

Internet of things is an idea that physical devices like sensors and actuators can be connected over the internet to pull data. In this chapter, we will be looking at how to use the Raspberry Pi for home automation and integrating services that we use daily. We will also discuss how to control our Raspberry Pi GPIO remotely from anywhere in the world over the internet.

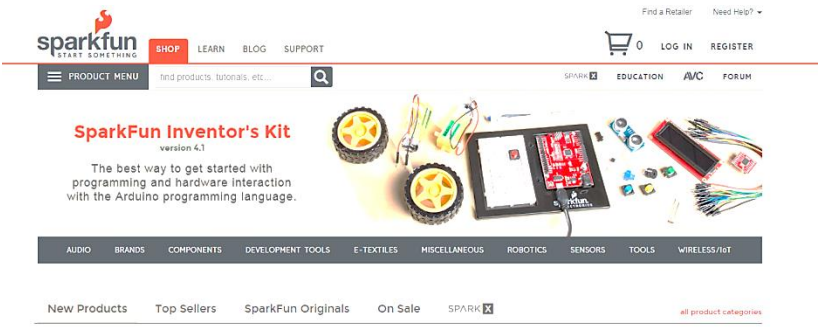
Internet of Things Services

One of the most important "internet of things" service providers you should consider is IFTTT, which stands for "If This, Then That." IFTTT links services together; in other words, if a section of event occurs on one service, IFTTT will create an action on another. For you to understand IFTTT better log into the IFTTT web site www.ifttt.com , scroll down to their discover section and click on any of the Applets. There are several Applets such as applets for following news, applets for android, applets for marketers, applets for iOS, applets for outer space, etc. These applets are developed and integrated with different handy services. Another important "Internet of Things" service provider is 'Particle." Particle is an IoT company that produces web-connected hardware which

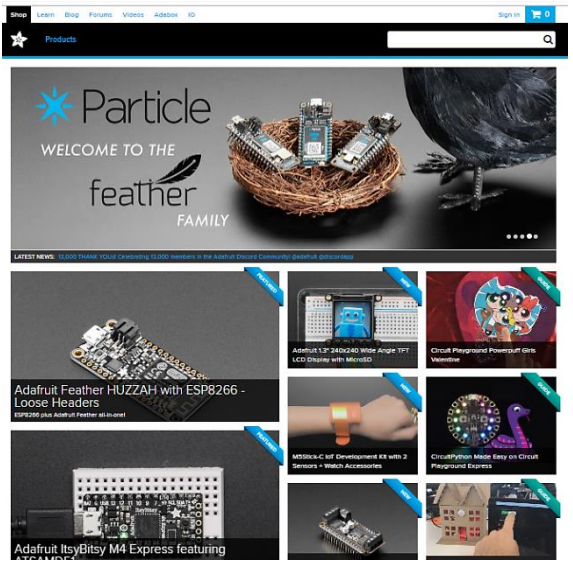
also supports Raspberry Pi. We can interact with the Raspberry Pi via the IoT service particle cloud. In other words, the Raspberry Pi can be programmed wirelessly and have it interact with other devices on the particle cloud. You can access the particle cloud via www.particle.io.



To get started, create an account with IFTTT and particle on their websites.



Another service you should sign up to for this project is a SparkFun account. You can create a Sparkfun account on their website data.sparkfun.com. Sparkfun is where you can push data to be stored and displayed. For instance, if you have temperature and humidity sensor on your Raspberry Pi, you can push that weather station data to this server and have it stored for you. Adafruit can provide a similar service. Adafruit have service called adafruit IoT which works as a publish and subscribe scheme called MQTT.



MQTT is a useful protocol for machines to communicate with each other. MQTT works by publishers and subscriptions. In other words, a publisher will publish data to some topic, and the data is stored on a web server. The web server analyzes the data, goes through those who subscribed to that topic, and pushes it to them. So, publishers publish to a topic, and any subscriber to that topic gets delivery of the data.

If you don't already have one, you will need to also create a Gmail account. If you already have a Gmail account, but you intend to keep your Internet of Things (IoT) project account separate from your personal Gmail account, you can create a new account for that purpose. So, in summary, create an IFTTT, particle, and Gmail account for the IoT projects we will be demonstrating in this chapter.

Controlling the Raspberry Pi Remotely Using IFTTT and Particle

In this section, we will demonstrate how we can remotely control our Raspberry Pi over the internet using our Gmail, IFTT, and particles accounts. Let's have an overview of how this thing works. First, we will connect our Gmail to IFTTT, we will then link the IFTTT to the particle cloud, and the particle cloud will publish events to our Raspberry Pi which is running a piece of particle software called particle agent.

Illustration: In this example, we will connect an LED to our Raspberry Pi and control the Power ON/OFF the LED from another location over the internet. Here are the steps.

- We will receive some email, and we will focus on the subject.
- We will change the state of the LED connected to our Raspberry Pi based on the content of the subject of the email we will receive.
- The content of the subject we will be looking out for is led-on/led-off, and this will be our control signal.

- IFTTT will look at the subject of the email and decide what to do about the content. If the content matches either 'led-on' or 'led-off,' then IFTTT will publish a particle cloud event which will be pushed to the Raspberry Pi.

Step 1: Open the terminal on your Raspberry Pi and type in the command to setup particle-agent on your Raspberry Pi `pi@devpi3b:- $ bash <(curl -sL https://particle.io/install-pi)` and execute that by hitting the enter key on the keyboard. This will download and install particle-agent.

Step 2: You will be prompted to enter your credentials for your particle account. This is the email address and password you used in signing up for the particle cloud.

Step 3: The Raspberry Pi logs into particle, and you have the opportunity to give your Raspberry Pi a name. This will be required for reference purposes; for this example, I would name my Raspberry Pi "smithpi3." Feel free to name yours with whatever you like and hit the enter key on the keyboard. Now the Pi is registered to the particle cloud, and the Raspberry Pi would start running a particle app called "Tinker"

If you want to roll back your particle app on your Raspberry Pi to factory default, you can always enter the command

```
Pi@devpi3b:- $ particle-agent setup
```

This will roll it back to where you have to sign in with your particle account and name your Pi.

Step 4: Connect a single LED on a breadboard to your Raspberry Pi.

Step 5: Over on your computer, go to build.particle.io. This will open up an online programming environment (web IDE). While that is opening, on another URL tab, open the IFTTT website and a third URL, open your Gmail account. Go back to the Particle IDE.

The particle IDE is where we will program our Raspberry Pi. On the left side of the IDE, you find some example apps, and on the right side is where we write our codes. At the lower right corner, you would see the device connected indicator, and by the side, you would find the name of your Raspberry Pi. This is an indication that the particle agent is connected and running on our Raspberry Pi. This means we will be programming our Raspberry Pi from within our web browser and we can do this from anywhere in the world where there is internet access.

Step 6: copy the Raspberry Pi particle-agent demonstration code and paste in the IDE or type it directly.

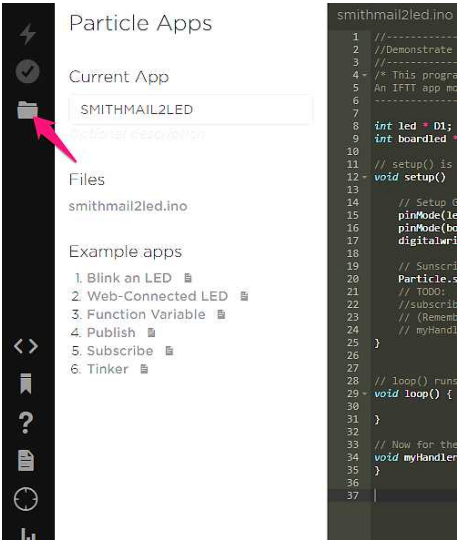
```
//---------------------------------------------------------------------------
//Demonstrate Particel and IFTTT
//-----------------------------------------------------
/* This program subscribes to a particular event.
An IFTT app monitors inbox activity of a Gmail account and publishers events to the Particular Cloud.
-----------------------------------------------*/

int led * D1;
int boardled * D7;

// setup() is run only once, it's where we set up GPIO and initialise peripherals
void setup()  {

    // Setup GPIO
    pinMode(led,OUTPUT); //Our LED is output (lighting up the LED)
    pinMode(boardLed,OUTPUT); // Our on-board LED is output as well
    digitalwrite(boardLed,LOW);

    // Sunscribe to an event published by IFTTT using Particle, subscribe
    Particle.subscribe("unique_event_name", myHandler);
    // TODO:
    //subscribe will listen fr the event, unique_event_name and when it find
    // (Remember to replace unique_event_name with an event name of your own choosing. Mak it somewhat complicated to make sure it is unique)
    // myHandler() is declared later in this app.
}


// loop() runs continously, it's our infinite loop. In this program we only
void loop() {

}

// Now for the myHandler function, which is called when the Particle cloud
void myHandler(const char *event, const char *data)
}

```

One thing you have to change is the unique_event_name. This is a public event, so we would try to write something really unique so that anyone publishing to this event with the correct data would be able to do what the code is meant to do. I would use something really long and unique to replace the unique_event_name. Let's use smith_is_expectng_this_mail_122132211

```
//---------------------------------------------------------------------------
//Demonstrate Particel and IFTTT
//-----------------------------------------------------
/* This program subscribes to a particular event.
An IFTT app monitors inbox activity of a Gmail account and publishers events to the Particular Cloud.
-----------------------------------------------*/

int led * D1;
int boardled * D7;

// setup() is run only once, it's where we set up GPIO and initialise peripherals
void setup()  {

    // Setup GPIO
    pinMode(led,OUTPUT); //Our LED is output (lighting up the LED)
    pinMode(boardLed,OUTPUT); // Our on-board LED is output as well
    digitalwrite(boardLed,LOW);

    // Sunscribe to an event published by IFTTT using Particle, subscribe
    Particle.subscribe("unique_event_name", myHandler);
    // TODO:
    //subscribe will listen fr the event, unique_event_name and when it find
    // (Remember to replace unique_event_name with an event name of your own choosing. Mak it somewhat complicated to make sure it is unique)
    // myHandler() is declared later in this app.
}


// loop() runs continously, it's our infinite loop. In this program we only
void loop() {

}

// Now for the myHandler function, which is called when the Particle cloud
void myHandler(const char *event, const char *data)
}

```

```
//---------------------------------------------------------------------------
//Demonstrate Particel and IFTTT
//-----------------------------------------------------
/* This program subscribes to a particular event.
An IFTT app monitors inbox activity of a Gmail account and publishers events to the Par
-----------------------------------------------*/

int led = D1;
int boardled = D7;

// setup() is run only once, it's where we set up GPIO and initialise peripherals
void setup()  {

    // Setup GPIO
    pinMode(led,OUTPUT); //Our LED is output (lighting up the LED)
    pinMode(boardLed,OUTPUT); // Our on-board LED is output as well
    digitalwrite(boardLed,LOW);

    // Sunscribe to an event published by IFTTT using Particle, subscribe
    Particle.subscribe("smith_is_expectng_this_mail_ 122132211", myHandler);
    // TODO:
    //subscribe will listen fr the event, unique_event_name and when it find
    // (Remember to replace unique_event_name with an event name of your own choosing.
    // myHandler() is declared later in this app.
}


// loop() runs continously, it's our infinite loop. In this program we only
void loop() {

}

// Now for the myHandler function, which is called when the Particle cloud
void myHandler(const char *event, const char *data)
}
```

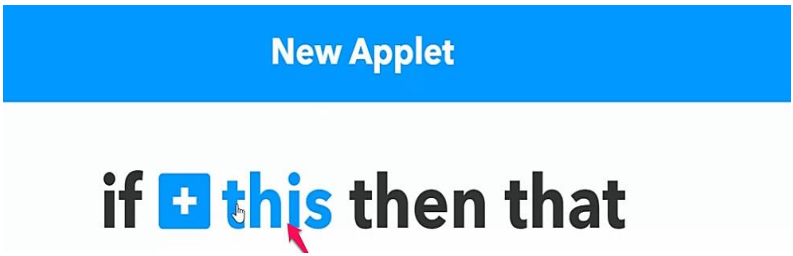
Step 7: We will now save our code with any name. SMITH MAIL2LED and click on "Save" Icon by the left and the name appears under "My Apps"



Now let's look at how the code in the particle script works and how it handles events. The Particle.subscribe is something that runs like a particle agent, and it subscribes a particle to an event. The event has a name, and whenever the name of the event is triggered, it triggers myHandler. myHandler is a function we write that takes in the event name and some data.

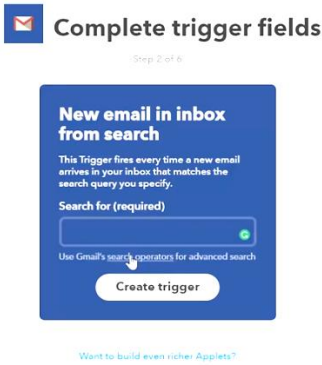
Step 8: Go over to IFTTT and log into your account. By your account name, on the top right corner, click on the arrow pointing down and click on "New applet"

Under the "New applet" window, click on "+ this" to add an 'if trigger.' This will display all the services to link to IFTTT.

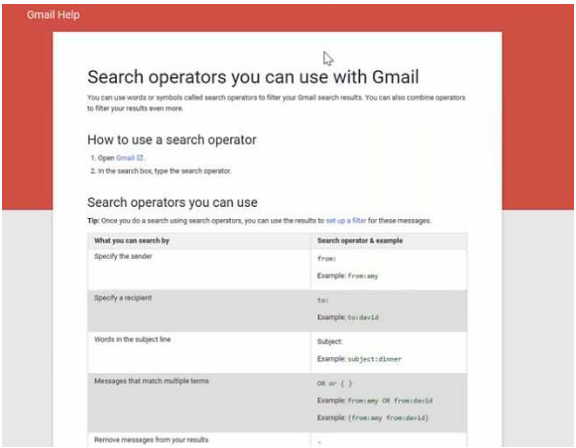


Search for "Gmail" using the search bar and click on it. You will find several Gmail trigger options like 'any new email in inbox', 'any new attachment in inbox', 'New email in inbox

from', 'New starred email in inbox', 'New email in inbox labeled' and 'New email in inbox from search'. Choose "New email in inbox from search." This gives us a lot of options to work with.



Click on the "Search operators" to find out more about this particular trigger.



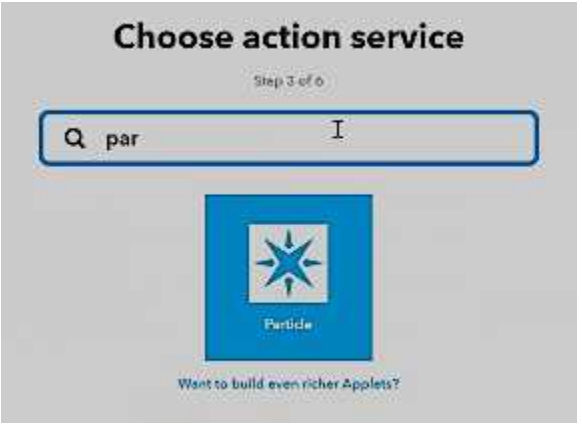
You will find several examples of operators you can use with Gmail. These are examples of how to write Gmail trigger syntax. Remember we are searching for a syntax that has to

do with the subject field. That will trigger LED-On and LED-Off. Scroll down to where it says "Word in the subject line– () Example: Subject: dinner." Type this word out in the 'search for (required)" box. So in the box type in Subject: led-on. Remember we want to trigger an event based on "led-on" and also a subject "led-off." We need to figure out how to do that. Click on the "Search operators" at the "Use Gmail's search operators for advanced search just above the "Create Trigger" button to look for a syntax that will handle this event. The best option here is "Messages that match multiple terms – OR or { } Example: from:any Or from:david, {from:any from:david}". This is like a "OR Statement" where we can use the word "OR" or wrap our syntax in a curly bracket.

So we create our trigger using both the "Word in the subject line" and "Messages that match multiple terms" syntax.

Finally, Click on "Create trigger." This will populate your 'If' section of the If then + that trigger with the Gmail logo. Next, we choose our action by click on "+ that" of the If then that" command. This takes us to the action service menu, where we can choose our action.

Our action is a particle action. In the search box, type "particle" and click on the magnifying glass icon. Click on the particle logo that pops up.



The particle action is where we will publish our event. So, click on the "Publish an Event" option.

In the "publish an event" dialogue box, we will key in the Event name, Data that should be included in the event and indicate if the event is a public or private event. This determines who can have access to the event.

The event name we are working is the unique event name from our particle IDE **`smith_is_expectng_this_mail_122132211.`**The data is what will be passed to the Raspberry Pi, and we want to pass the subject which is led-on and led-off. So leave the "The event includes (Data) section empty and makes the event a "Public" event. Click on the "Create action" and "Finish" button. This is all we need to do to get our email-to-LED functionality working.

Step 9: We can now perform our real test. Go over to your Gmail account, clickon "Compose," for the address, write your email address. This means you are mailing your self with the subject led-on—and click on the "Send" button. IFTTT services can take several minutes to come through while you are debugging. You can go back to IFTTT where you created your app and click on "Check Now" to carry out a manual update.

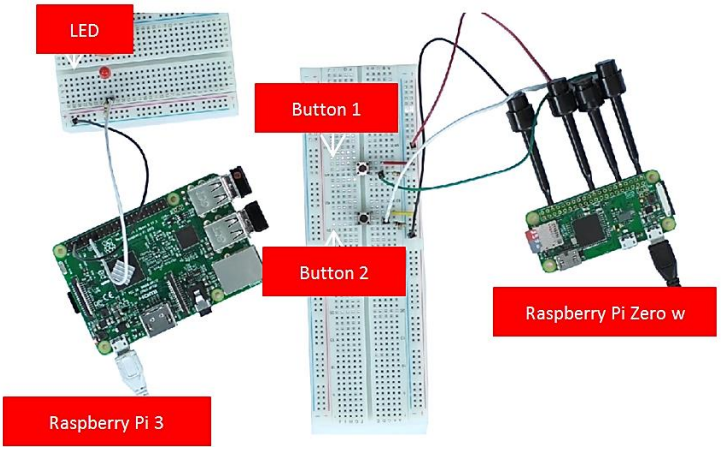
The LED on the breadboard comes On. Go back to "Compose," send an email to yourself again with the subject led-off. Go back to IFTTT and click "Check now" and this should power off the LED on the breadboard.

Controlling the Raspberry pi remotely using Only the Particle Cloud

In the last section, we saw how we could link our particle account with a Gmail account using IFTTT. This is good for some interesting but complicated projects that integrate a lot of different services. But the problem is that IFTTT can be a little bit slow sometimes. It can take several minutes for an IFTTT trigger to be processed—probably because it is a free service. If you are using your Raspberry for an application where timing is important and you want to respond very quickly to an event, then using IFTTT is not an ideal way to go. You can keep your entire projects only within the particle cloud and publish between several Raspberry Pi or other particle devices.

Illustration: In this demonstration, we will be using two Raspberry Pi. A Raspberry Pi 3 and a Raspberry Pi Zero W. The Raspberry Pi Zero W is good for small power-efficient enclosed projects, that is, it can be installed somewhere, e.g., a remote sensor. In this project, we will connect two buttons to the Raspberry Pi Zero W, and a LED will be connected to the Raspberry Pi 3. When we press the first button, it will flash the LED on a breadboard connected to a Raspberry 3 remotely. So on the first press; the LED will flash once, on the second press it will flash twice, on the third press it will flash thrice and so on. The second button is a reset button that resets the total count. Even though both Raspberry Pi's are

sitting on the same table, they are not connected physically except through the particle cloud.



We would now write the code for this demonstration in the particle IDE(build.particle.io/build/new). From our first example we attached our Pi to our particle IDE by setting up a particle agent on our Raspberry 3 using the `pi@devpi3b:- $ bash <(curl -sL https://particle.io/install-pi)` command. We will also do that for the Raspberry Pi Zero W and register it to our particle account as "PiZeroW."

The first piece of code will be for the publisher (Raspberry Pi Zero W), and the second piece of code is for the subscriber (Raspberry Pi 3).

To write the publisher code, on the particle IDE dashboard, click on the "Device" icon at the lower-left corner.

This will display the devices registered to your particle account. Select the publisher device, which is the Raspberry Pi Zero W and click on the "flash" button at the upper left part of the particle IDE dashboard.

When the flashing is done, type in the subscriber code below in the IDE

```
//-------------------------------------------
// Publisher code line
/*-------------------------------------------
        Each time a button is pressed, publish the total number of
        Button-pressed to an event.
-------------------------------------------------------------
// GPIO Definitions
int pwr3_3 = D12; //3.3V supply for button
int flashButton = D13;
int resetButton = D16;
int boardLed = D7;
int pressCount = 0; // Stores number of times flash-button is pressed


// Setup GPIO
Void setup()  {
        pinMode(flashButton, INPUT);
        pinMode(resetButton, INPUT);
        pinMode(boardLed, OUTPUT);
        pinMode(pwr3_3, OUTPUT);

        digitalWrite(boardLed,HIGH); //Start the onboard LED as OFF
        digitalWrite(pwr3_3, HIGH);
}

// poll the button status, increment or reset the counter,publish the
count
Void loop() {
   if (digitalRead(flashButton) == HIGH)
        pressCount++;
        String datastring = String(pressCount);

        Particle.publish("buttonPressed",dataString,60,PRIVATE);
        // And flash the on-board LED on and off.
        digitalwrite(boardLed,LOW);  //On
        delay(500)
        digitalwrite(boardLed,HIGH);  //Off

  }


if (digitalRead(resetButton) == HIGH) pressCount = 0; //Resets
                                                        the counter
```

For this illustration, let's save the code in the particle IDE as 'PUB1", you can save yours using any name. Next, click on the "CREATE NEW APP" button on the particle IDE dashboard. For this illustration, I will name it SUB1 for 'subscribe' and write the subscribe code below into the IDE.

```
//-----------------------------------------------
// Subscriber code line
/*-----------------------------------------------
      Subscribe to an event and assume all data that comes in is
      Numeric. Convert the string data to an integer and flash an
        LED that number of times
-------------------------------------------------------------------

int led = D1;
volatile int blinkCount = 0;
volatile int bool trigger = 0;

Void setup()  {

        //Setup GPIO
      pinMode(led, OUTPUT); // Our LED pin is output(light up LED)
      digitalwrite(led,LOW);

        // subscribe to an event published by the second pi
      Particle.subscribe("buttonPressed", myHandler, MY_DEVICES);
        // The MY_DEVICES argument means only events published by devices
        registered to this particle cloud account will be accepted

    }

   void loop() {
        if (trigger) {// trigger is set True in the event handler
           for(int i = 0; I < blinkCount; i++){
              digitalWrite(led, HIGH);
              delay(350);
              digitalWrite(led, LOW);
              delay(350);
       }
       trigger = 0; // Need to reset the trigger
     }
}

void myHandler(const char *event, const char  *data)
{
    String dataVar = data;  //Load data into a string var
    blinkCount = dataVar.toInt();
    trigger = 1; // Trigger the blink loop

}
```

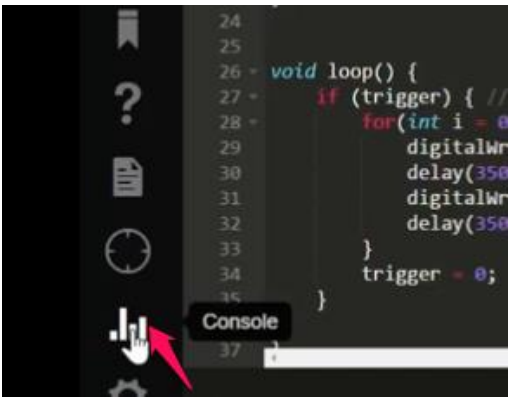
Set the code to the Raspberry Pi 3 by clicking on the name you saved it with, then save the project and "flash" the device. From our code, you can see that the subscriber

`Particle.subscribe("buttonPressed", myHandler, MY_DEVICES);`

Is subscribed to a non-unique event name called "button-Pressed" which is general-purpose but you can see that we have got this extra argument MY_DEVICES for the subscribe option. What we are doing here is restricting the instances of this event to only come from MY_DEVICES. These are devices that are registered only to your account. This way, you can use much simpler event names because you are making it a private event only within your account.

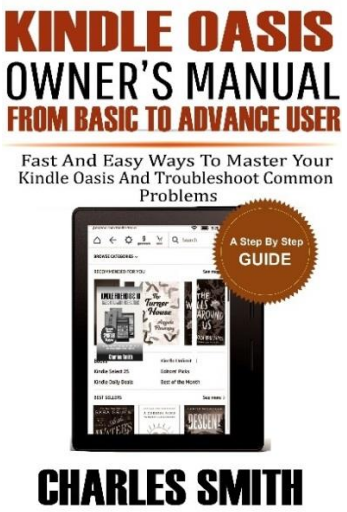
Now when we press the button 2 on the breadboard, we will get the LED on the other breadboard to flash once, if we press it again, the LED will flash twice, etc. If we press the button 1, it resets, and if we press the button2 again, the LED flashes once and continue to flash depending on the number of times we press the button.

We can see what is going on behind the scene by clicking on the console button to open up the particle IDE Console.

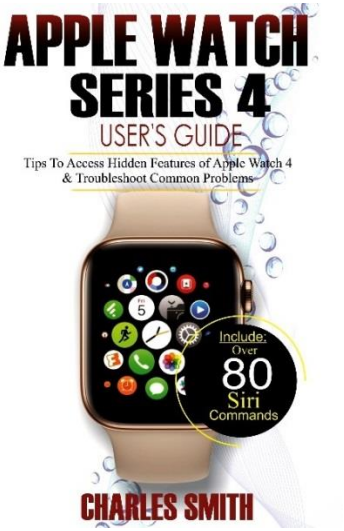


This shows you a log of events as well as a timeline. It displays the event name, date, and time the event was published and the names of devices involved.

Books by the Author



https://www.amazon.com/dp/B07DH7FK3B



https://www.amazon.com/dp/B07L5PRYXX

https://www.amazon.com/dp/B07L5PRYXX