*Introduction to UI based computer control*

# ARDUINO + PYTHON
## PROGRAMMING FOR ROBOTS

NARESH KUMAR T & THILEEPAN S

*Arduino + Python*

*Programming for Robots*

# ARDUINO + PYTHON
# PROGRAMMING FOR ROBOTS

NARESH KUMAR T & THILEEPAN S

# CONTENTS

# ABOUT THE BOOK

This book is about creating interactive GUI –Graphical User interface projects with help of Python and Arduino. Advancements in Open Source computer programming and electronics has led to the growth of STEM education, DIY (Do-it-Yourself) activities and maker's movement. The world of robotics is crowded with various hardware and software platforms with different types of communication protocols. The easy way to learn Robotics is to start with the right hardware and software. Arduino is a powerful, opensource embedded hardware used widely in the field of robotics, automation and IOT. Python is a opensource powerful programming language used world wide by programmer's, developer's and DIY enthusiast's. This book is a step by step guide for the students to get started with hands-on-experience with coding and control of robots.

## WHAT WILL YOU LEARN

- Familiarize with the Python Tkinter GUI widgets
- Interfacing and controlling Arduino
- Write code with both Python and Arduino
- Get examples of real world application
- Learn by doing - Challenges/ Capstone Project

# FOR THE READERS

*Arduino & Python programming for Robots* is intended for anyone who already know a basic of Python and/or who is learning Python and wish to integrate Arduino hardware control with a graphic user interface to their applications. This book will serve as a useful guide for intermediate Python or Arduino programmers to develop user interface based control system for their Robotic applications. Over all you get to understand the real world application with an example and exercises filled with events, function calls and Serial

## ABOUT THE AUTHORS

***Naresh Kumar Thanigaivel*** is a a Front-end Product developer and digital artist and holds a Master's degree in Mechanical Engineering. He has 3 Years of Work experience in both academic research and Industrial domain in Bio-inspired robotics & Consumer electronics field. He aims to fuse his creativity and innovation in User-Centric design to improve the way of human interaction with Products. He loves to design and fabricate robots, develop hardware and digital Illustrations as his passion.

***Thileepan Stalin*** is a passionate Engineer with master's degree in Mechatronics. Previously, he worked as an Assistant Professor for 3 Years in India. Now, he is building team for his dream company, Team DNA Robotics. He developed various robots, interactive games, software applications for desktop and mobile phones. He also authored a book on "Computer Vision with OpenCV and Python 3: Practical examples workbook".

In this book, we are going to learn about the familiarizing of python programming, creating Graphical User Interface, Communicating with embedded hardware and software. Though, Robotics is like an Ocean, In this book, we tailored the experiments to give a foundation for the beginner's. All the codes are tested and it works well. In case of any difficulties, Don't panic, try to understand the code and try to rewrite and align the code based on the

## ACKNOWLEDGEMENT

We would like to acknowledge Python and Arduino for their open source data and examples. We dedicate this book to all open source enthusiasts and people who inspired us to write this book.

# 1

# Getting started with Arduino

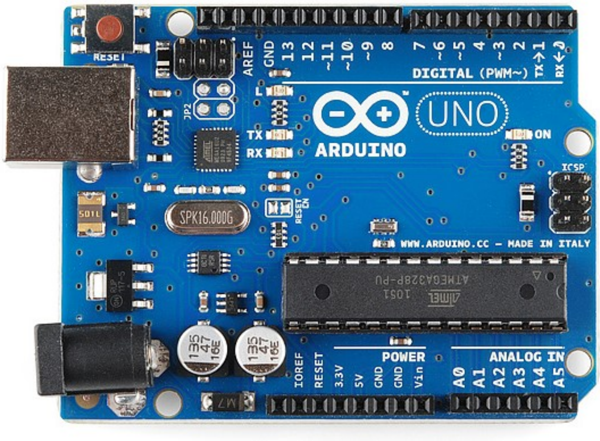
**This chapter will let you get started with Arduino and steps required to set up arduino IDE (Integrated development environment) on your PC. Then, we will use an example program to upload into Arduino microcontroller.**

## 1.1. WHAT IS ARDUINO?

Arduino is an open-source platform used for rapid prototyping of electronics projects. Arduino programming requires a physical programmable microcontroller board and an Arduino IDE (Integrated Development Environment) that runs on your computer. Arduino IDE is useful in writing control programs and uploading the code to the physical board.

Arduino platform is easy to learn and widely used by electronics beginners and hobbyist so it is very popular. Previous programmable boards, needs a separate hardware (programmer) for uploading the code into board. In Arduino, you can upload a new code with your USB without any additional hardware. Arduino programming is easy because of it's simplified C++ syntax and active documentation and blogs of open source community. It has a wide range of libraries and support for various hardware's (Sensors, LED's,  motors) and breakout

Arduino Uno
Micro-controller



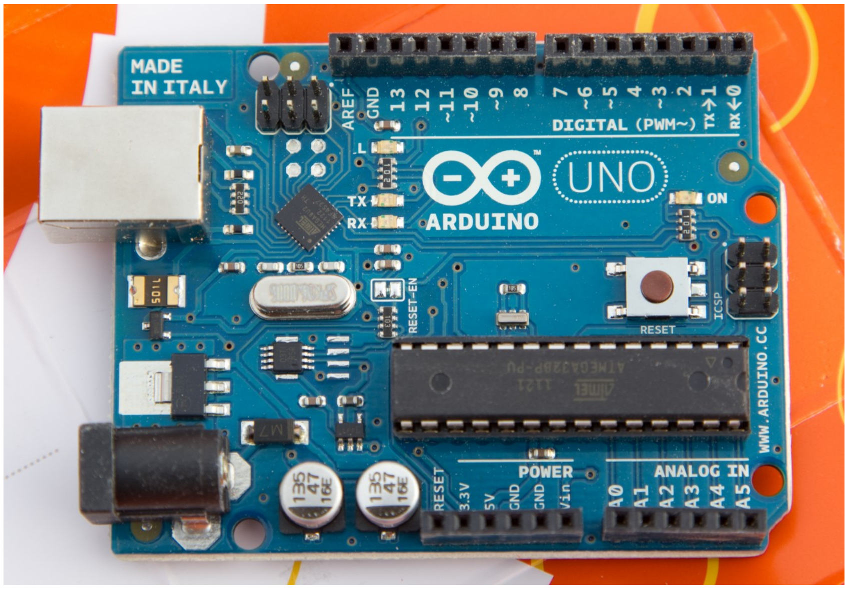
## 1.2. HOW DO I PROGRAM AN ARDUINO?

The Arduino IDE, open source software available on multiple platforms: Windows, Mac, and Linux.  There is a offline version of Arduino IDE which works on your laptop/ computer (without the need for internet) and web-based version which works on the browser (needs internet). The IDE helps you to write code faster with syntax and to upload the code to the microcontroller board in a simple button click. There are few VPL (Visual Programming Language) softwares like Scratch for Arduino, ArduBlock and Minibloq which makes Arduino programming easier for Kids. In this book, we will use Arduino IDE offline version for all our programs and

## 1.3. INSTALL ARDUINO IDE

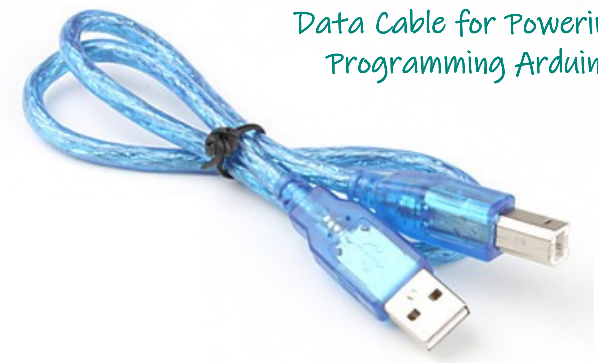
**Required Materials**

- A computer (Windows, Mac, or Linux)
- An Arduino-compatible microcontroller (Prefer Arduino UNO if you are beginner)
- A USB A-to-B cable, or another appropriate way to connect your Arduino-compatible microcontroller to your computer



Arduino Uno

Micro-controller

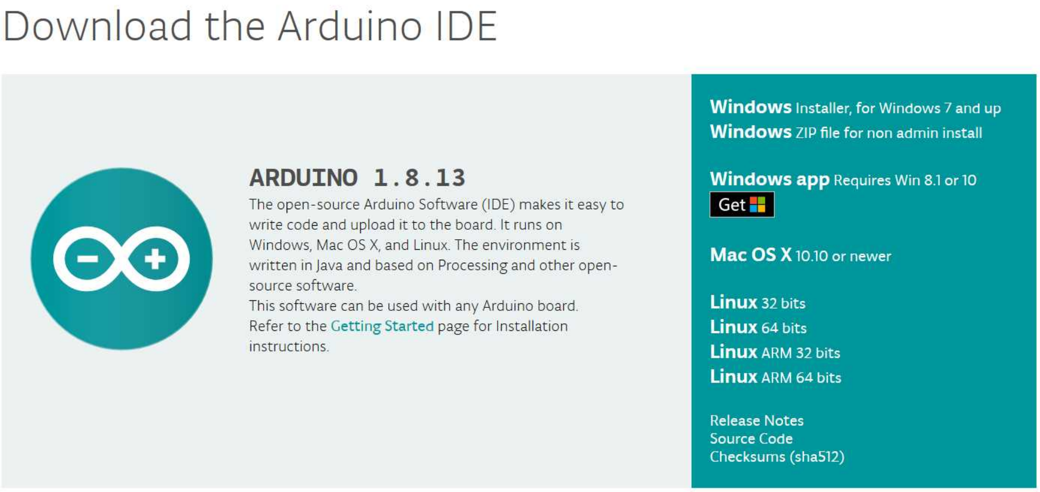


Data Cable for Powering / Programming Arduino

Notebook / Work station with WINDOWS / MAC OS installed

## Download the Arduino Software (IDE)

Get the latest version from the download page. (https://www.arduino.cc/en/Main/Software) You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. You can also manually install the Zip package if you are comfortable with.



Download the Arduino IDE

**ARDUINO 1.8.13**

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

**Windows** Installer, for Windows 7 and up
**Windows** ZIP file for non admin install

**Windows app** Requires Win 8.1 or 10
Get

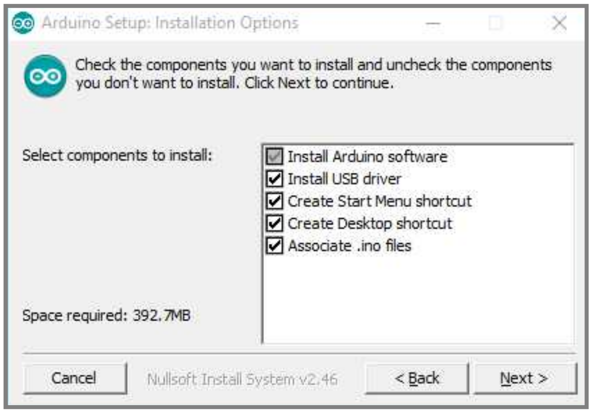
**Mac OS X** 10.10 or newer

**Linux** 32 bits
**Linux** 64 bits
**Linux** ARM 32 bits
**Linux** ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

Find your installer here

When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system. Then Choose the components to install. Choose the installation directory (we suggest to keep the default one).



Arduino Setup: Installation Options

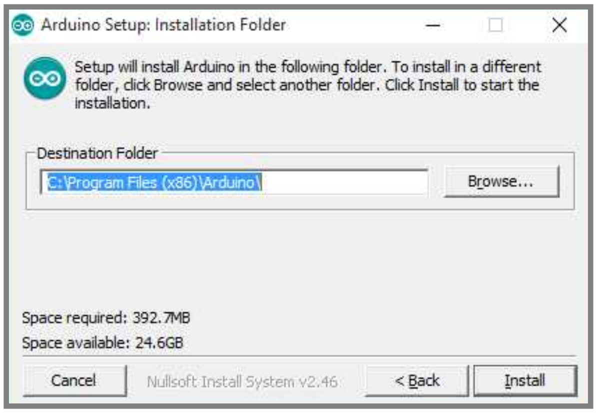
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

☑ Install Arduino software
☑ Install USB driver
☑ Create Start Menu shortcut
☑ Create Desktop shortcut
☑ Associate .ino files

Space required: 392.7MB

Select all and Click "NEXT"



Arduino Setup: Installation Folder

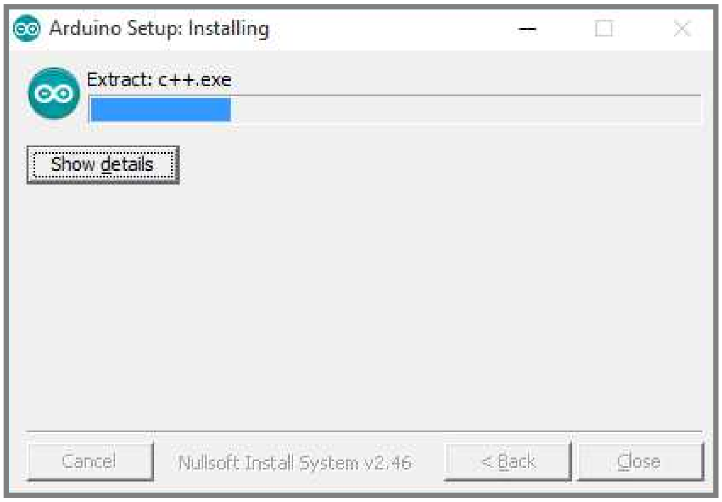
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino\

Space required: 392.7MB
Space available: 24.6GB

Leave the default location to 'C: drive' and click 'INSTALL'

The process will extract and install all the required files to execute properly the Arduino Software (IDE)

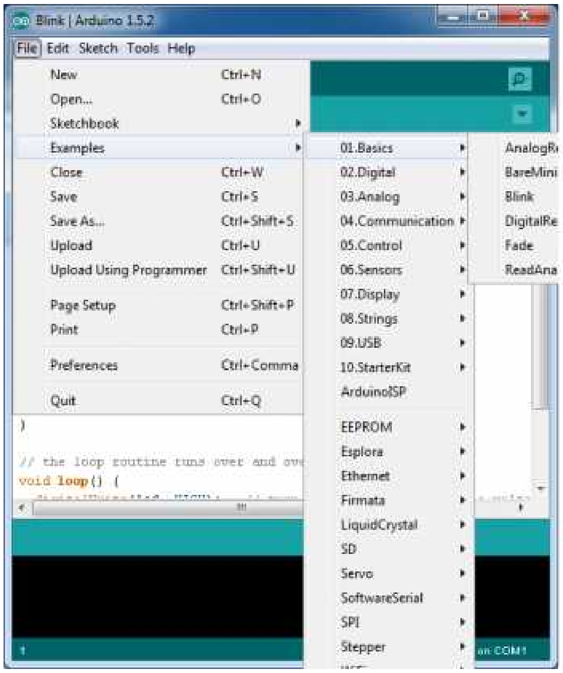


Give some time to complete the installation. Grab a cup of Coffee

## 1.4. LAUNCH & UPLOAD YOUR FIRST PROGRAM

After following the appropriate steps for your software install, we are now ready to test your first program with your Arduino board!

- Launch the Arduino application
- If you disconnected your board, plug it back in
- Open the Blink example sketch by going to: `File > Examples > 1.Basics > Blink`

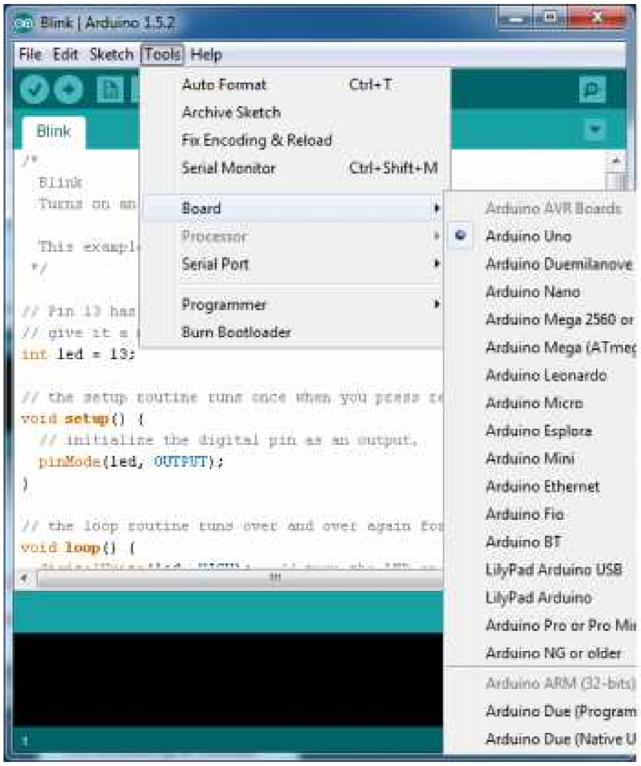


```
}

// the loop routine runs over and ove
void loop() {
```

Double Click on Arduino Icon on your desktop or in your START menu to open the IDE

Then Open the "BLINK" code from example section

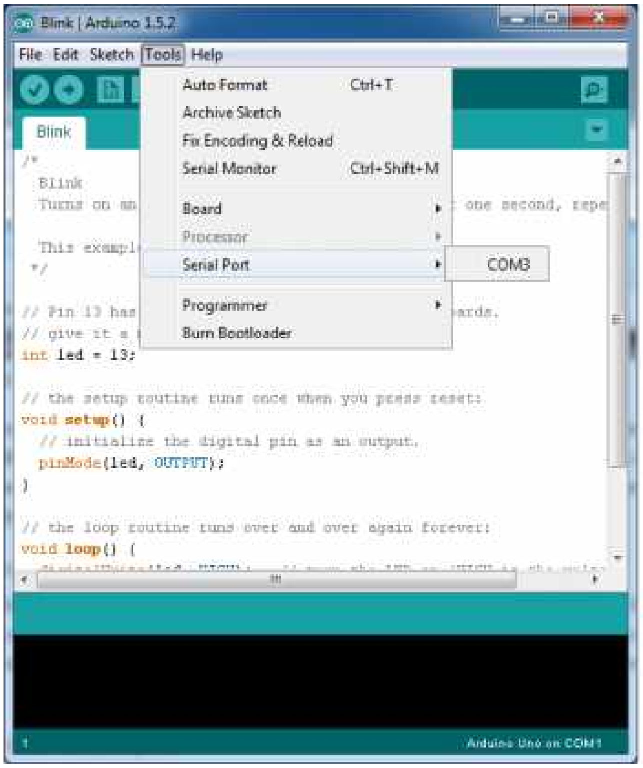
Select the type of Arduino board you're using: Tools > Board > your board type



```
/*
  Blink
  Turns on an

  This exampl
 */

// Pin 13 has
// give it a
int led = 13;

// the setup routine runs once when you press r
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again for
void loop() {
```

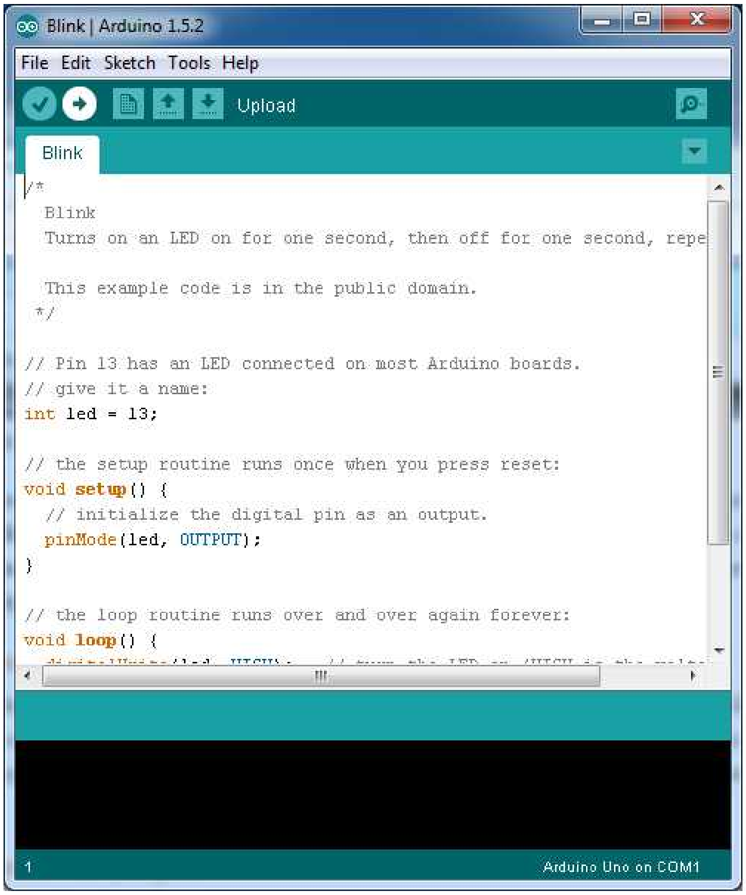
Select the serial/COM port that your Arduino is attached to: Tools > Port > COMX

Make sure to check the COM PORT every time before you upload your code



```
/*
  Blink
  Turns on an                              one second, repe

  This exampl
 */

// Pin 13 has                              oards.
// give it a
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

- If you're not sure which serial device is your Arduino, take a look at the available ports, then unplug your Arduino and look again. The one that disappeared is your Arduino.
- With your Arduino board connected, and the Blink sketch open, press the 'Upload' button



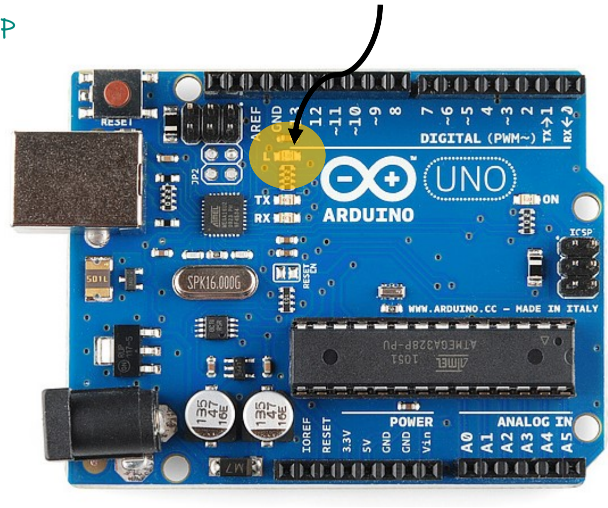
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
```

Your first program to try out. Upload the Program to Arduino by clicking "UPLOAD" button

- After a second, you should see some LEDs flashing on your Arduino, followed by the message 'Done Uploading' in the status bar of the Blink sketch.
- If everything worked, the onboard LED on your Arduino should now be blinking! You just programmed your first Arduino!

Remember to Connect the Arduino to laptop using USB Cable

On-board LED starts blinking



**Well done!**

# 2 Introduction to Python

**This chapter will guide you through setting up Python and installing Spyder IDE on your computer. Then you can check your python version and install suitable Python modules required for our experiments.**

## 2.1. PYTHON

Python, a popular programming language is widely used for software development, scripting and web development (server side). It was developed in 1991 by Guido van Rossum. Python has a simple syntax and works on multiple platforms (Windows, Mac, Linux, Raspberry Pi, etc). Python's syntax allows developers to write programs with countable lines of code. As the Python runs on a interpreter system, the code can be executed faster. In simple terms, Python is simple, faster and powerful to write complex programs in few lines of code.

## 2.2. INSTALLING PYTHON

**Install Python 3.6**

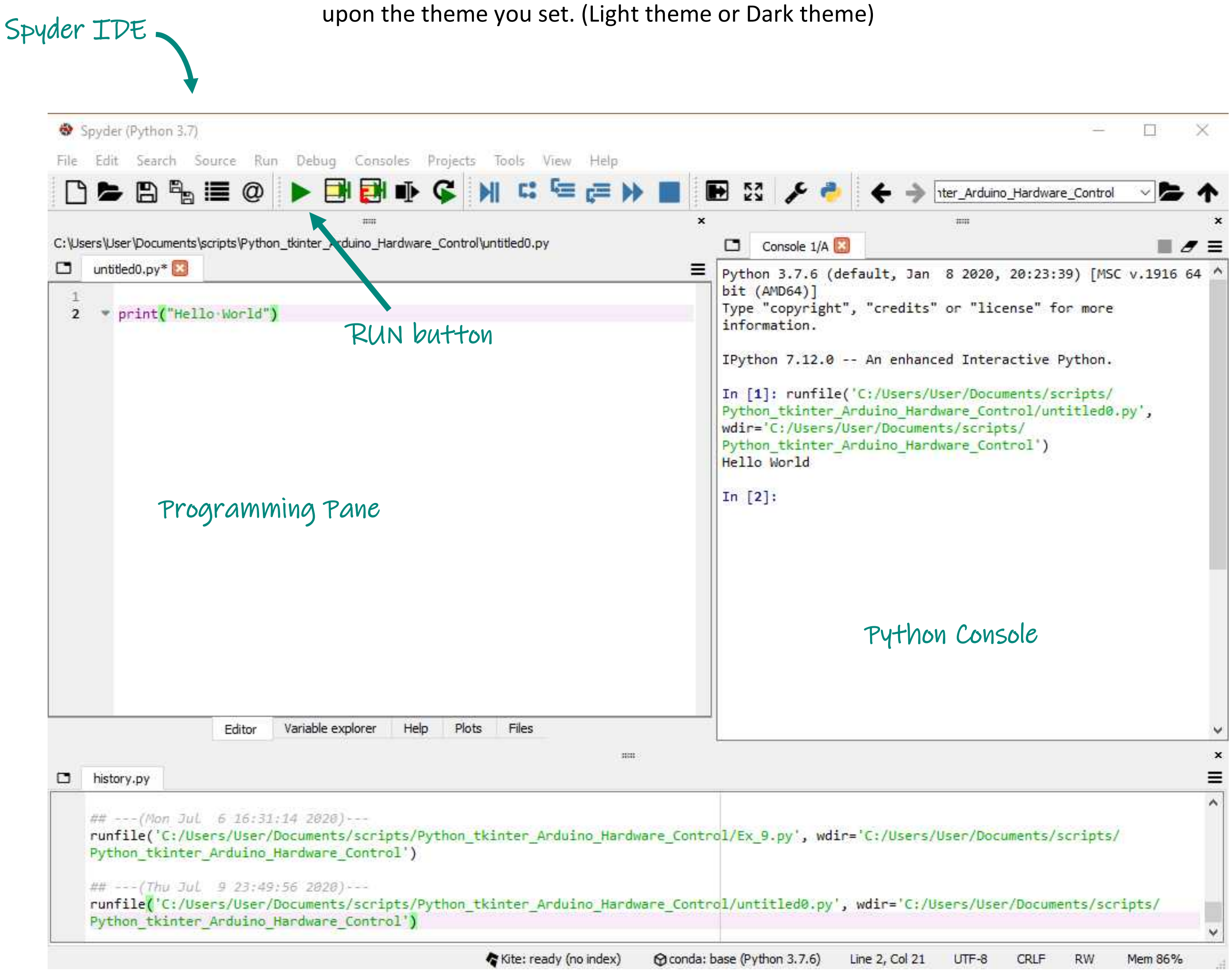
**Step 1:** Go to https://www.anaconda.com/download/

**Step 2:** Download Python 3.7 version (Select 64 bit or 32 bit version based on your system)

**Step 3:** Install Python 3.7 by running the downloaded file.

### SPYDER IDE

Go to START, Search for **SPYDER (anaconda 3)** and open it! It looks similar to below depending upon the theme you set. (Light theme or Dark theme)


Spyder IDE
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
ter_Arduino_Hardware_Control
RUN button
Programming Pane
Python Console
untitled0.py*
Console 1/A
Editor Variable explorer Help Plots Files
history.py
Kite: ready (no index) conda: base (Python 3.7.6) Line 2, Col 21 UTF-8 CRLF RW Mem 86%

C:\Users\User\Documents\scripts\Python_tkinter_Arduino_Hardware_Control\untitled0.py

```
print("Hello World")
```

```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64
bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/User/Documents/scripts/
Python_tkinter_Arduino_Hardware_Control/untitled0.py',
wdir='C:/Users/User/Documents/scripts/
Python_tkinter_Arduino_Hardware_Control')
Hello World

In [2]:
```

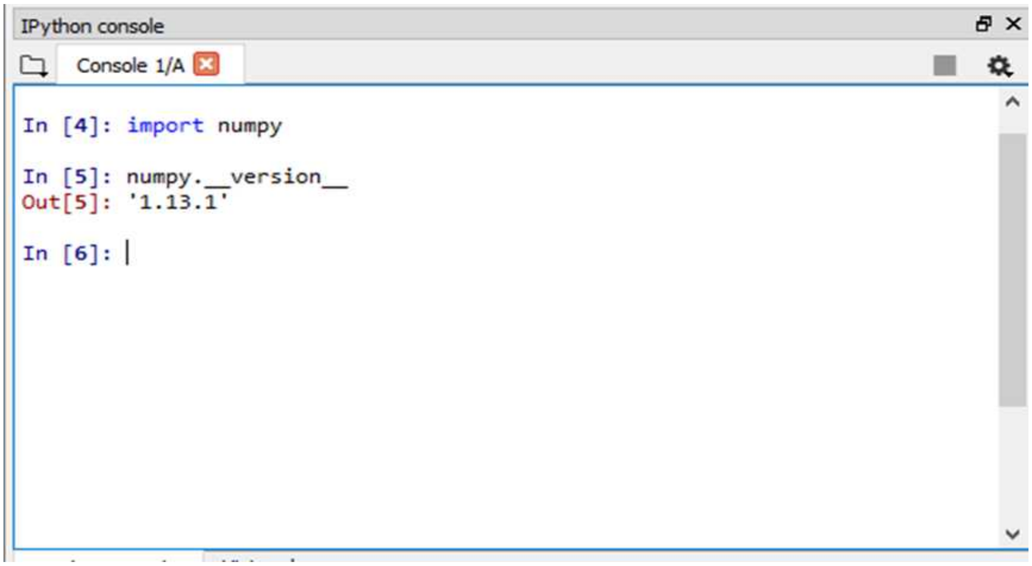
```
## ---(Mon Jul  6 16:31:14 2020)---
runfile('C:/Users/User/Documents/scripts/Python_tkinter_Arduino_Hardware_Control/Ex_9.py', wdir='C:/Users/User/Documents/scripts/
Python_tkinter_Arduino_Hardware_Control')

## ---(Thu Jul  9 23:49:56 2020)---
runfile('C:/Users/User/Documents/scripts/Python_tkinter_Arduino_Hardware_Control/untitled0.py', wdir='C:/Users/User/Documents/scripts/
Python_tkinter_Arduino_Hardware_Control')
```

## 2.3. TEST THE INSTALLED VERSION

Type the following in the Python Console

It will display the version of Numpy installed. Numpy is a numerical python module to work on numbers and basic math operations.

```
Import numpy

numpy.__version__
```



```
In [4]: import numpy

In [5]: numpy.__version__
Out[5]: '1.13.1'

In [6]: |
```

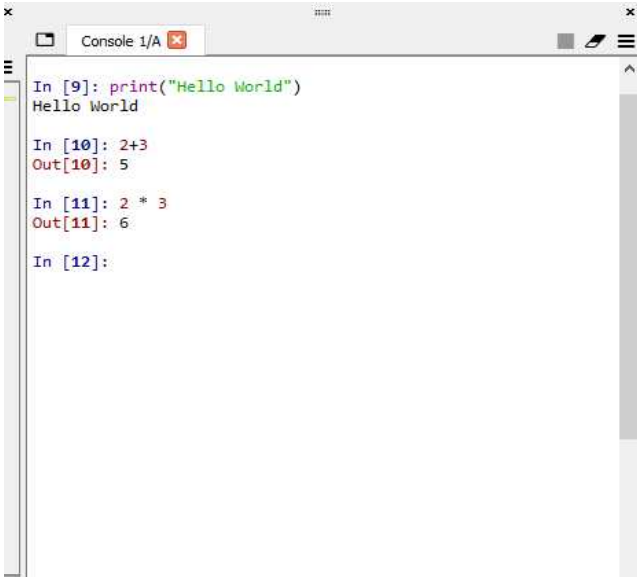
Python Console of Spyder IDE

## 2.4. FIRST PYHTON PROGRAM

Type the following in the Python Console

```
print("Hello World!")
```

It will print the "Hello World!" in the Python console.



```
In [9]: print("Hello World")
Hello World

In [10]: 2+3
Out[10]: 5

In [11]: 2 * 3
Out[11]: 6

In [12]:
```

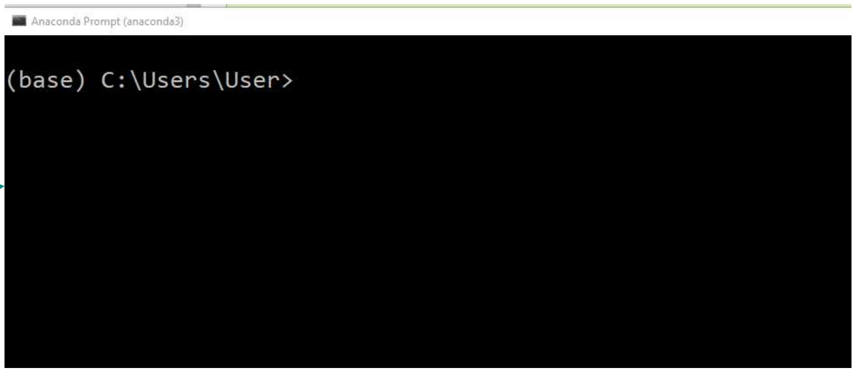
Try adding numbers.

Type 2 + 3

and click enter

## 2.5. INSTALLING PYTHON MODULES

In the upcoming exercises, we will get introduced to various Python modules that is useful for creating Graphical user interfaces and making Serial communication with external hardware. So, lets have a short overview of each modules and the procedures to install it. At first, lets open Anaconda command prompt shell. Go to Start, Search and open Anaconda Prompt.



```
(base) C:\Users\User>
```

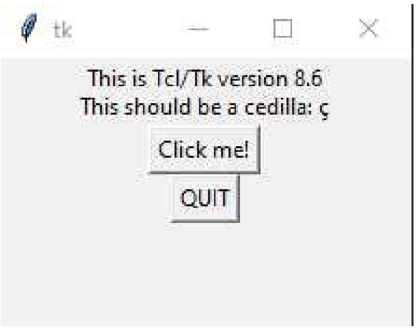
Anaconda command Prompt

### 2.5.1. TKINTER

The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, as well as on Windows systems. Tkinter is installed along with Pythion installation and no other setup is required. To check tkinter installation, run the following command in Anaconda command prompt shell.

```
python -m tkinter
```

This should open a window demonstrating a simple Tk interface and letting you know that tkinter is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.



This is Tcl/Tk version 8.6
This should be a cedilla: ç

Output

**Usage:**

```
>> import tkinter
```

Or

```
>> from tkinter import *
```

In the case, your tkinter is not installed, use the following command in Anaconda Prompt to install Tkinter.

```
for installation in pip and windows, run:
>> pip install tk
```

## 2.7. TK TOOLS

The tk_tools package has a collection of widgets and tools for efficient and easy creation of GUI elements.  These can be classified into three  types of widgets:

There are three categories of widgets:

- groups of widgets that are useful as a group
- visual aids using the canvas
- useful improvements on existing widgets

We will learn more in next chapter.

```
for installation in pip and windows, run:
>> pip install tk-tools


Usage:
>> import tk_tools
```

## 2.8. PYSERIAL

This module is useful in accessing the serial port using Python programming. The module named “serial” takes care of the necessary process involved

```
for installation in pip and windows, run:
>> pip install pyserial


Usage:
>> import serial
```

### 2.9. TK COLOR PICKER

This module contains a ColorPicker class which implements the color picker and an askcolor function that displays the color picker and returns the chosen color in rgb and html formats.

```
for installation in pip and windows, run:
>> pip install tkcolorpicker

Usage:
>> from tkcolorpicker import askcolor
```

### 2.10. PYFIRMATA

pyFirmata is a Python interface for the Firmata protocol.  pyFirmata is a useful package for working with arduino microcontrollers with direct access to input and output pins of Arduino. There is no need to program your Arduino every time. This gives flexibility and reuse of hardware for multiple projects without reprogramming it.

```
for installation in pip and windows, run:
>> pip install pyfirmata

Usage:
>> from pyfirmata import Arduino, util
```

We have the required modules installed.

Let’s jump into Tkinter now >

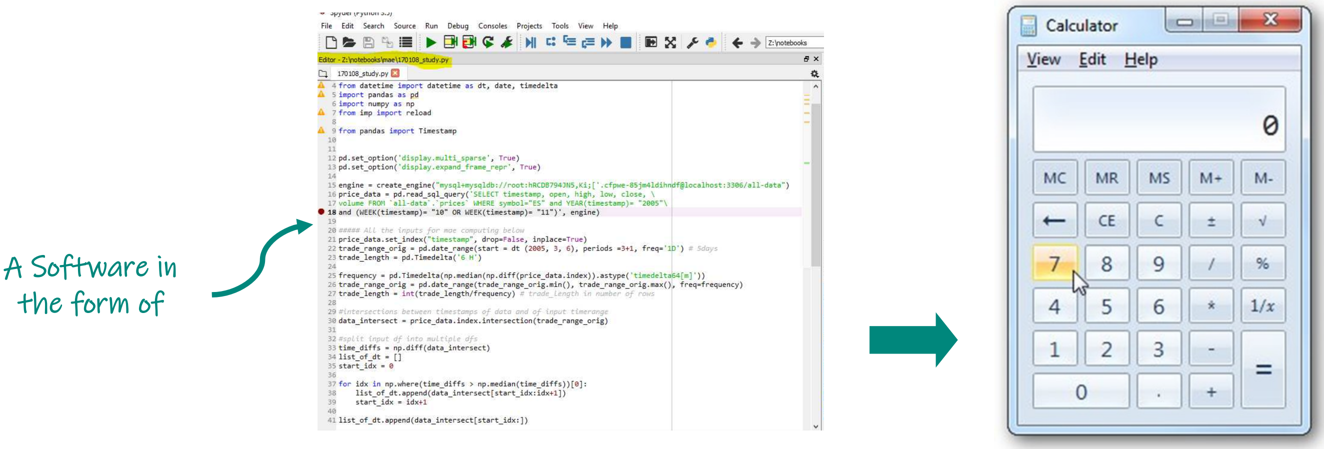
# 3 GUI Programming Python Tkinter

**This chapter takes you through python's tkinter module for creating basic GUI widgets and performing event calls over it. You will learn to arrange various widgets in a defined canvas area, apply colors, fonts and assign functions based on your design needs.**

## 3.1. Graphic User Interface (GUI)

A ***Graphic user interface (GUI)*** is a user based interaction that contains various elements from buttons, icons, windows, labels, switches in digital form. It lets users to interact, send commands and perform operations to a connected system. Without an user interface, it wouldn’t be intuitive to work with a computer program or a hardware system. So, a GUI is an integrated communication interface for users to interact with computers or hardware. We can also call a GUI to be a gateway for ***Human to computer interaction (HMI).***

Many of us write algorithms in a form of code using a IDE (Integrated development environment) and compile the code to get an output. Everything from compiling and visualizing your output happens with in IDE application. What If you want to share your code to other users? What if other users don’t have the IDE or knowledge about codes? So, here comes a need to develop a GUI and make it easy for users to make use of your application. It has to be fancy and institutive to use. A great examples of a simple GUI is a ***Calculator* app** that lets users to input numbers and perform mathematical operations and get an output displayed.



A Software in the form of

```
from datetime import datetime as dt, date, timedelta
import pandas as pd
import numpy as np
from imp import reload

from pandas import Timestamp


pd.set_option('display.multi_sparse', True)
pd.set_option('display.expand_frame_repr', True)

engine = create_engine("mysql+mysqldb://root:hRCDB794JN5,Ki;['.cfpwe-85jm4ldihndf@localhost:3306/all-data")
price_data = pd.read_sql_query('SELECT timestamp, open, high, low, close, \
volume FROM `all-data`.`prices` WHERE symbol="ES" and YEAR(timestamp)= "2005"\
and (WEEK(timestamp)= "10" OR WEEK(timestamp)= "11")', engine)

##### All the inputs for mae computing below
price_data.set_index("timestamp", drop=False, inplace=True)
trade_range_orig = pd.date_range(start = dt (2005, 3, 6), periods =3+1, freq='1D') # 5days
trade_length = pd.Timedelta('6 H')

frequency = pd.Timedelta(np.median(np.diff(price_data.index)).astype('timedelta64[m]'))
trade_range_orig = pd.date_range(trade_range_orig.min(), trade_range_orig.max(), freq=frequency)
trade_length = int(trade_length/frequency) # trade_length in number of rows

#intersections between timestamps of data and of input timerange
data_intersect = price_data.index.intersection(trade_range_orig)

#split input df into multiple dfs
time_diffs = np.diff(data_intersect)
list_of_dt = []
start_idx = 0

for idx in np.where(time_diffs > np.median(time_diffs))[0]:
    list_of_dt.append(data_intersect[start_idx:idx+1])
    start_idx = idx+1

list_of_dt.append(data_intersect[start_idx:])
```

***Software in “code” form Vs GUI form***

A GUI not only lets you to do basic functions like addition or subtraction on digital interface but also lets you to control complex hardware interfaces such as drones, humanoid robots and self-driving cars. In these applications, your GUI will look simple in its design and outlook but it can perform complex functions at a click of a button.  You can show a video feed from a camera fixed over your drone, make analysis of video captured and even store them to a specific folder of your need. You can monitor, log your sensor’s data and even watch the sensor’s data in real time graphs. A GUI’s design and application is countless and tied to designers needs. So, lets get started to create GUI’s using python modules.

## 3.2. PYTHON’s GUI FRAMEWORKS

There are different libraries available in Python to create GUI. Each of them has it’s pros and cons. We will discuss few of them to get an overview:

*TKinter*

**Tkinter** is Python’s standard GUI framework which comes integrated with Python. It is simple to use and available under Python open-source license.

*PyQT*

**PyQt** is a multi-platform targeted python bindings implementing the Qt library for Qt application development framework. PyQT gives rich UI elements/widgets similar to desktop apps. It gives the freedom for the developer to create program with python and to use Qt Designer to create visual UI elements.

*Kivy*

**Kivy** is an open-source Python library for developing multi-touch apps with polished UI. Kivy supports almost all platforms like Windows, MacOSX, Linux, Android, iOS. Kivy is best suited for developing GUI’s for touch screen based mobile devices and tablets.

*WxPython*

**WxPython**, also known as WxWindows (WxWidgets library), is an open-source abstract-level wrapper for cross-platform GUI library. It is implemented as a Python expansion module. With WxPython, a programmer can develop native applications for multiple OS.

*PyGUI*

**PyGUI** is a graphical application cross-platform framework for Unix, Macintosh, and Windows. PyGUI is a simple and lightweight platform which can display UI elements with few lines of code. It is useful for native application development.

*PyForms*

**Pyforms** is a cross-environment framework for developing GUI applications. The framework offers a Python layer of desktop forms, based on PyQt, OpenGL and other libraries. This is useful for applications to run Desktop GUI, Web and terminal without requiring code modifications. Pyforms promotes modular software design and code reusability with minimal effort.

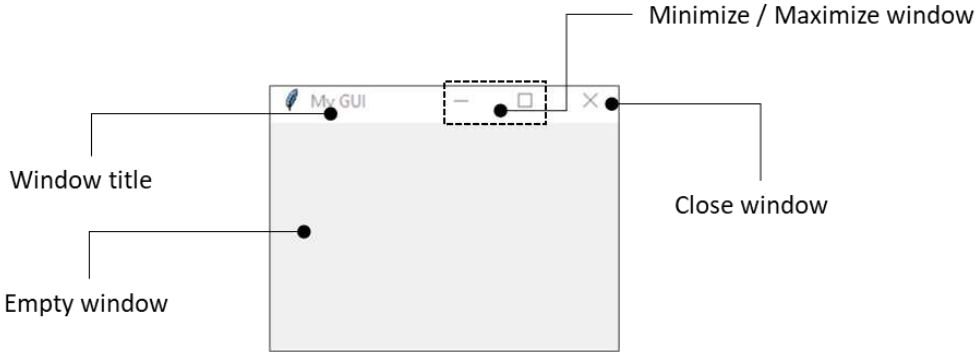
In this book, we will focus on Tkinter module based GUI development. It is simple and has tools to interface with third party API’s. By learning to use Tkinter module in creating GUI widget’s and arranging the layout of UI elements, the user has an edge to explore other GUI development modules. With this foundation of Tkinter knowledge, a user can do advanced GUI development.

## 3.3. TKINTER WINDOW

Let's get started with creating a window by importing ***Tkinter*** package in python. Then we initialize a window manager using ***tkinter.Tk()*** method. You can assign this to a any variable name. In this exercise, lets assign it to variable called *window.* This method creates a blank window with options to close, maximize, and minimize. You can then give this window a title using *window.**title("your_title")*** method. Finally, let's use the ***mainloop()*** method to display the window until you manually close it. It runs an infinite loop in the backend. And, when you execute the program, it displays the output in a separate window.

**Python**

```python
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.mainloop()
```



If you get the output as depicted above, Hurrah! Well done.

You can even fix the size of window using *window.**geometry()*** method and by specifying the required dimensions in pixels. If you are detailed person and want to assign a specific color to window,  *window.**configure()*** method allows you to assign background color for your window.

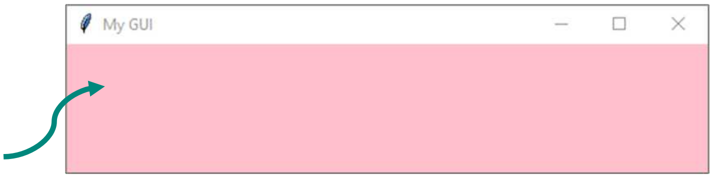
Go ahead and add the following code before the mainloop() to see the changes.

*Color names includes:*

**"red"**
**"orange"**
**"yellow"**
**"green"**
**"blue"**
**"purple"**

**Python**

```python
window.configure(background="pink")
window.geometry("500x100")
```



You can assign any color

## 3.4. TKINTER WIDGETS

Now that we created a empty window for our GUI, its time add widgets to it. We can call these widgets to be elements of our GUI. Every element has its own properties and configurations that can be defined according to our need. Lets understand each of the widget and use them in our GUI.

### BUTTON

A **`Button`** widget can be named and used to trigger an event whenever its clicked. Its basically a on/off switch assigned to a specific event. A button can be configured to add a text, shape, color and even control its state to act ad '*normal*' or '*disabled*'. (In a '*disabled*' state, you cannot click the button)

**Syntax:** `button_widget = tkinter.Button(window, options=value )`

where window is the argument for the parent window while option is a placeholder that can have various values like foreground & background Color, font, command (for function call), image, height, and width of button.

Once you define the button, the next step is to arrange it to your window. There are few methods to arrange the widgets on to your window provided by geometry manager. They are,

**`pack():`** It organizes the widgets in a block manner, and the complete available width is occupied by it. It's a conventional method to show the widgets in the window.

**`grid():`** a grid is another way to organize the widgets. It uses the Matrix row-column concept. Grid primarily takes two parameters: row and columns as shown in example below.

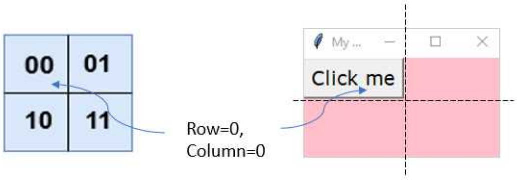
**`place():`** It helps to place the widgets at a specific position as instructed by the user in the parent widget.

For our learning, we will only use the **`grid`** method to arrange our widgets in desired location. Since its based on rows & columns like a table, we can specify them relatively simple way. The following example will help you to create a button widget and arrange them into your window.

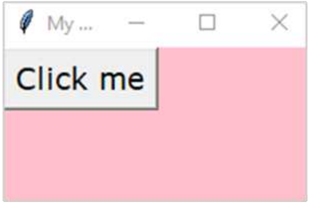
Try Changing the button state to be "*tkinter.DISABLED*" and see what happens to your button.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="pink")
window.geometry("200x100")
button = tkinter.Button(window, text = "Click me",
                         font = ('Verdana',14), bg ="white", state=
tkinter.NORMAL)
button.grid(row=0, column=0)
window.mainloop()
```



| | |
|---|---|
| 00 | 01 |
| 10 | 11 |



## LABEL

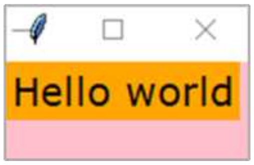
A *label* is a useful widget to indicate a function about other widgets in our GUI. A label can also be used to show images if required. We can call labels to be a place holder.

**Syntax:** `label_widget = tkinter.Label(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like the text, font type, font size, background color, image, width, and height of button.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="pink")
window.geometry("100x50")
label = tkinter.Label(window, text = "Hello world",
                    font = ('Verdana',14), bg ="orange")
label.grid(row=0, column=0)
window.mainloop()
```



Hello world

Output

## ENTRY

A **`Entry`** widget is used to create input fields or to get input text from the user within the GUI.

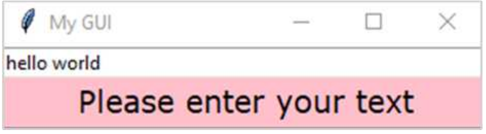
**Syntax:** `entry_widget = tkinter.Entry(window, option= value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="pink")
window.geometry("300x50")
label = tkinter.Label(window, text = "Please enter your text",
                    font = ('Verdana',14), bg='pink')
label.grid(row=1, column=0)
entry = tkinter.Entry(fg="black", bg="white", width=50)
entry.grid(row=0, column=0)
window.mainloop()
```

Output



hello world

Please enter your text

Now you've got some text entered into the **Entry** widget, but that text hasn't been sent to your program yet. You can use **.get()** to retrieve this text and assign it to a variable of your own.

```
Python
```

```
text_input = entry.get()
print(text_input)
```

```
>>> hello world
```

## CHECK BUTTON

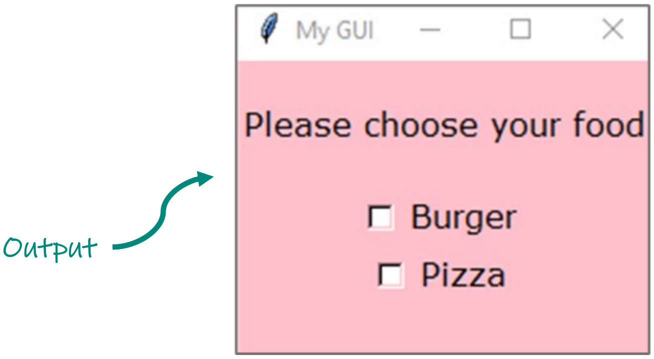
A **Check Button** presents the user a set of predefined options in the form of boxes. The user is allowed to select more than one available option.

**Syntax:** `Check_Button = tkinter.Checkbutton(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

```
Python
```

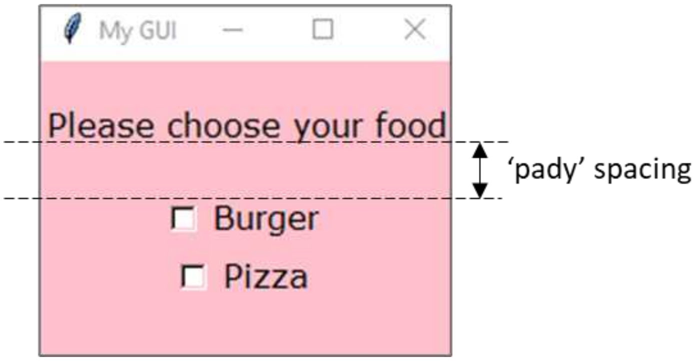
```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("210x150")

label = tkinter.Label(window, text = "Please choose your food",
                      font = ('Verdana',12), bg='white')
label.grid(row=0, column=0, pady=20)

Check_button1 = tkinter.Checkbutton(window, width = 15,text= "Burger",
                                    font = ('Verdana',12), bg='white')
Check_button1.grid(row=1, column=0)

Check_button2 = tkinter.Checkbutton(window, width = 15,text= "Pizza",
                                    font = ('Verdana',12), bg='white')
Check_button2.grid(row=2, column=0)
window.mainloop()
```



**Note:**

In this Check button GUI, we used a option called “pady” for arranging the label in our window. This option allows you to add space between your widgets in x and y direction. You can use *padx*, *pady* option and specify the spacings in pixel values. A quick example is as follows:

```
label.grid(row=0, column=0, padx=10, pady=20)
```



## RADIO BUTTON

A **`radio button`** is a Tkinter GUI widget that allows the user to choose only one of a predefined set of mutually exclusive options.

**Syntax:** `Radio_Button = tkinter.Radiobutton(window, option=value)`

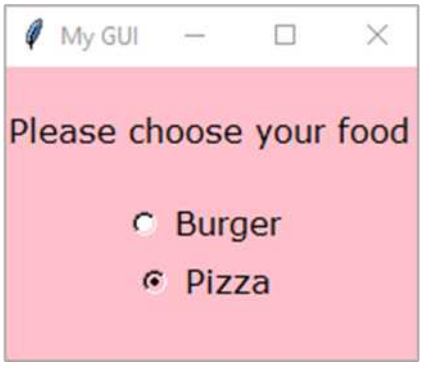
where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```python
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("210x150")
label = tkinter.Label(window, text = "Please choose your food",
                   font = ('Verdana',12), bg='pink')
label.grid(row=0, column=0, pady=20)
```

**Python**

```
Radio_button1 = tkinter.Radiobutton(window, width = 15,text= "Burger",
                          font = ('Verdana',12), bg='pink', value = 0)
Radio_button1.grid(row=1, column=0)

Radio_button2 = tkinter.Radiobutton(window, width = 15,text = "Pizza",
                                 font = ('Verdana',12), bg='pink',
value = 1)
Radio_button2.grid(row=2, column=0)
window.mainloop()
```



Please choose your food

○ Burger

● Pizza

Output

## LIST BOX

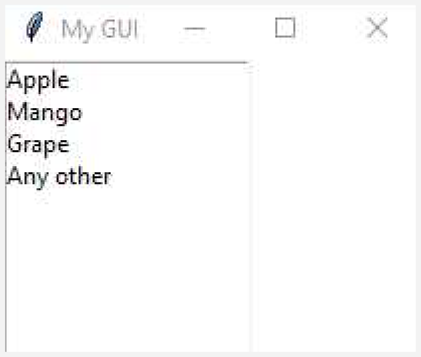
A **list box** offers a list to the user from which the user can accept any number of options.

Syntax: `list_box = tkinter.Listbox(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("210x150")
List_box = tkinter.Listbox(window)
List_box.insert(1, 'Apple')
List_box.insert(2, 'Mango')
List_box.insert(3, 'Grape')
List_box.insert(4, 'Any other')
List_box.grid(row=0, column=0)
window.mainloop()
```



```
Apple
Mango
Grape
Any other
```

Output

## FRAME

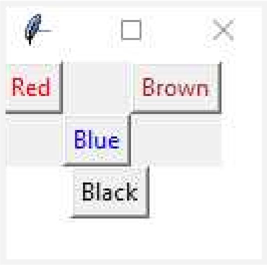
**`Frame`** acts as a place holder to hold the widgets. It is used for grouping and organizing the widgets independent of window. A frame can be very useful while adding many widgets into window where certain elements should be independent on other elements position.

Syntax: `frame = tkinter.Frame(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder

**Python**

```python
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("100x100")
frame = tkinter.Frame(window)
frame.grid(row=0, column=0)
bottomframe = tkinter.Frame(window)
bottomframe.grid(row=1, column=0)
redbutton = tkinter.Button(frame, text = 'Red', fg ='red')
redbutton.grid(row=0, column=0)
greenbutton = tkinter.Button(frame, text = 'Brown', fg='brown')
greenbutton.grid(row=0, column=3)
bluebutton = tkinter.Button(frame, text ='Blue', fg ='blue')
bluebutton.grid(row=1, column=2)
blackbutton = tkinter.Button(bottomframe, text ='Black', fg ='black')
blackbutton.grid(row=0, column=0)
window.mainloop()
```



Output

## CANVAS

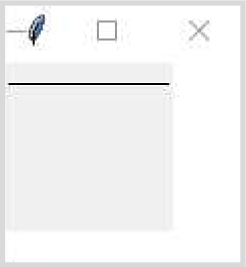
A **canvas** is very useful for drawing images and other graphics in our tkinter window. It is also useful in adding complex widgets in our window.

Syntax: `canvas = tkinter.Canvas(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("100x100")
canva = tkinter.Canvas(window, width=80, height=80)
canva.grid()
canvas_height=20
canvas_width=200
y = int(canvas_height / 2)
canva.create_line(0, y, canvas_width, y )
window.mainloop()
```



Output

## MENU BUTTON

A **Menu Button** lists down top down options in the top of window. It always stays on the top of window and useful for creating a like to various functions of your GUI.

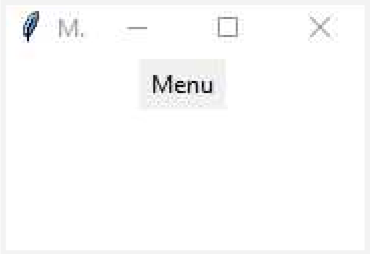
Syntax: `menu = tkinter.MenuButton(window, option=value)`

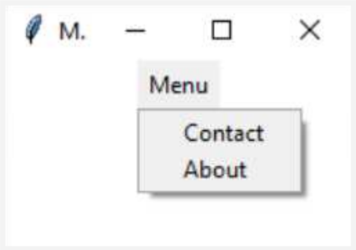
where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

Python

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("100x100")
mb =  tkinter.Menubutton ( window, text= "Menu")
mb.grid()
mb.menu  =  tkinter.Menu ( mb, tearoff = 0 )
mb["menu"] =  mb.menu
a  = tkinter.IntVar()
b = tkinter.IntVar()
mb.menu.add_checkbutton ( label ='Contact', variable = a)
mb.menu.add_checkbutton ( label = 'About', variable = b)
mb.pack()
window.mainloop()
```

Output





## MESSAGE BOX

A **Message Box** is similar to label widget for showing texts/messages whenever needed.

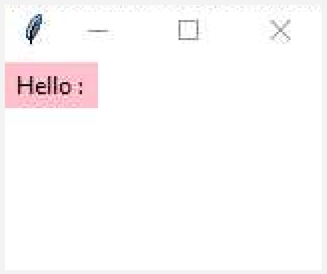
Syntax: `menu = tkinter.MessageBox(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

Python

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
window.geometry("100x100")
ourMessage ='This is our Message'
messageVar = tkinter.Message(window, text = "Hello:")
messageVar.config(bg='pink')
messageVar.grid(row =0, column=0 )
window.mainloop()
```

Output



Hello :

## SCALE

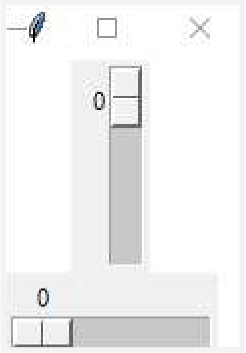
A **Scale** provides a graphical slider that allows to select any value from that scale. You can assign the limits of the scale, orientation and length/width of scale as well.

Syntax: `menu = tkinter.Scale(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
scale = tkinter.Scale(window, from_=0, to=50)
scale.grid()
scale2 = tkinter.Scale(window, from_=0, to=100, ori-
ent=tkinter.HORIZONTAL)
scale2.grid()
window.mainloop()
```



Output

## SCROLL BAR

A **Scroll bar** provides slide controller to slide long all available options, texts etc. Its much like a scroll bar in MS Word or website.

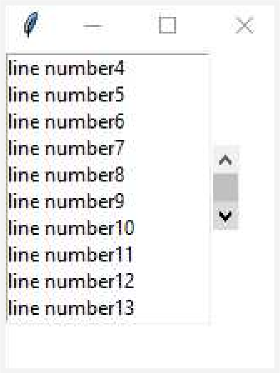
Syntax: `menu = tkinter.Scrollbar(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
scrollbar = tkinter.Scrollbar(window)
scrollbar.grid(row=0,column=1)
mylist = tkinter.Listbox(window, yscrollcommand = scrollbar.set )
for line in range(50):
   mylist.insert(tkinter.END, 'line number' + str(line))
mylist.grid(row=0,column=0)
scrollbar.config( command = mylist.yview )
window.mainloop()
```

Output



```
line number4
line number5
line number6
line number7
line number8
line number9
line number10
line number11
line number12
line number13
```

## TEXT

A **Text widget** provides the user to add a multi-line text and format to required style accordingly.

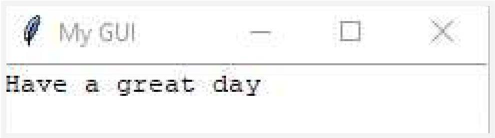
Syntax: `menu = tkinter.Text(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
T = tkinter.Text(window, height=2, width=30)
T.grid()
T.insert(tkinter.END, 'Have a great day\n')
window.mainloop()
```

Output



```
Have a great day
```

## SPIN BOX

A **spin box** is type of entry widgets where the inputs are defined by a limits. User can able to spin between multiple inputs to choose from.

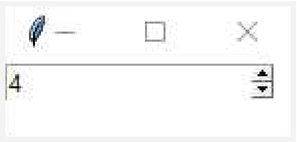
Syntax: `menu = tkinter.Spinbox(window, option=value)`

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
window.configure(background="white")
box = tkinter.Spinbox(window, from_ = 0, to = 10)
box.grid()
window.mainloop()
```

Output



4

## MESSAGE BOX

A **`Alert / message box`** provides the user to create a pop up altert message based on an event. It can be a click of a button or a notification of time or anything.

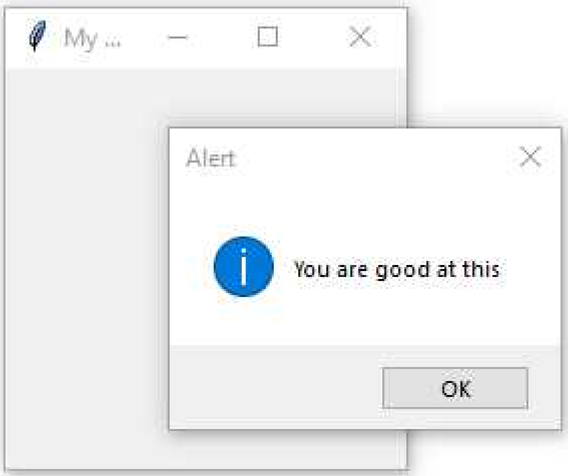
Syntax: alert = tkinter.messagebox.showinfo(“alert box name”, “your message”)

where window is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height of button etc.

**Python**

```
import tkinter
window = tkinter.Tk()
window.title("My GUI")
alert = tkinter.messagebox.showinfo("Alert", "You are good at this")
window.mainloop()
```

Output



You are good at this

**Note:**

You can use multiple types of alter / message boxes such as,

`Showinfo()` : Displays a info message

`showerror()` : Displays error

`showwarning()`: for displying warning

`Askquestion()`: Asks a yes or no question

Usage example:

```
alert = tkinter.messagebox.showwarning("Alert", "Warning! Stop this
action")
```

## 3.5. ADDING IMAGES TO TKINTER WIDGET USING PIL-

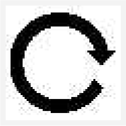
***Pillow*** is a python imaging library (PIL) used for handling images. We can open, manipulate, save images using this module. We make use of Pillow package to add images to our tkinters widgets. You can import JPG or PNG images of your wish and display it in tkinter canvas or buttons or labels.

**Usage:** `from PIL import Image`

The following code lets you to assign an image to a variable and open an image.

Python

```
from PIL import Image

right = Image.open("right.png") #please change  the file path to your directory

right.show()
```



Output

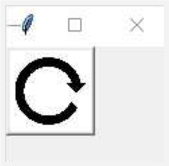
Now let’s try binding the same image to a tkinter button. You can modify the size of your image according to button dimension. A good way is to resize in external application such as *Paint.*

But before we begin, to insert an image into tkinter window, we need make use of ImageTK module. The ***ImageTk*** module contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images. Try the following code to display a image on tkinter’s button.

**Usage:** `from PIL import ImageTk`

Python

```
from PIL import Image, ImageTk

import tkinter

window = tkinter.Tk()

window.title("My GUI")

right = ImageTk.PhotoImage(Image.open("right.png"))

button = tkinter.Button(window, image = right)

button.grid(row=0, column=0)

window.mainloop()
```



Picture inside Tkinter window

If you get “***PyImage doesn’t exist***” error after you Run the code, Please restart your kernel Or Close your Spyder and Open again.

## 3.6. EVENT CALLS WITH TKINTER WIDGETS

So far, we have created various tkinter widgets, configured them and packed them in desired position on out tkinter window. All the widgets meant to do its function. For example, if a button is clicked, it has to give us an output. This output can be in the form of printed messages or can even send a data to external application. Let's begin with a basic event call with an example below.  We will be using the ***command*** option of tkinter widgets to connect a function. So, we also need to define a function call for this. If your never defined a function in python, no worries. We will cover this here.

In the following example, let us create a button and label widget and add events to them. Whenever the "ON" button is clicked, the label is updated to a defined text "Lights is ON". Simultaneously, if the user clicks "OFF" button, the label is updated to "Lights is OFF" text. Isn't cool! Lets try out.

**Python**

Import tkinter module

```
import tkinter
```

Function definitions

```
def lights_on():
    label.configure(text='Lights is ON', font= ('Verdana',16))

def lights_off():
    label.configure(text='Lights is OFF', font= ('Verdana',16))
```

Initialize tkinter window

```
window = tkinter.Tk() #create tkinter window
window.title("Events with buttons") #give title
window.configure(background="white") #change background color
```

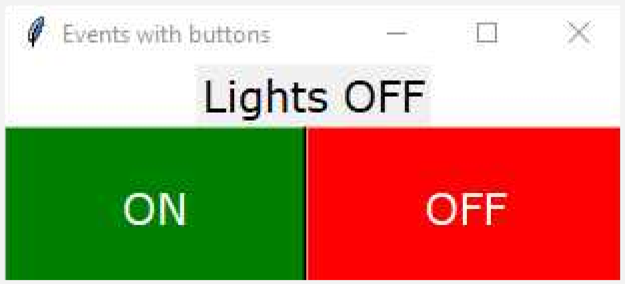
Buttons with commands and paddings to arrange properly

```
button_on = tkinter.Button(window, text="ON",
                           font= ('Verdana',16), padx=50, pady =20,
                           bg="green",fg="white",
                           command = lights_on) #Command links to function
button_on.grid(row=1,column=1)

button_off = tkinter.Button(window, text="OFF",
                            font=('Verdana',16), padx=50, pady =20,
                            bg="red",fg="white",
                            command = lights_off) #Command links to function
button_off.grid(row=1,column=2)
```

Define label widget

```
label = tkinter.Label(window, text="Lights OFF", font= ('Verdana',16))
label.grid(row=0, column=1, columnspan=3)
window.mainloop()
```

Output



**Task:**

Its seems the event function works as programmed.

But did you notice the buttons never change their state as seen in physical switches in our daily life? What do you think should be added to our button configuration to achieve this?

Refer to *Button widget* to add a state of NORMAL & DISABLED to button function call.

**Usage:** ***button.configure(state= tkinter.DISABLED)***

We will cover this in our exercise 1 if you can't solve it.

Still not satisfied!

Let's give a try to another GUI to add 2 numbers together. We will define a addition ***function*** and ***return*** the values to a text widget. This time we are performing a mathematical function by getting ***inputs from users*** and showing ***outputs to a text widget.***

**Python**

```
import tkinter
def add():
    x = int(in1.get())  #get value from entry box
    y = int(in2.get())  #get value from entry box
    z = x + y
    label.configure(text= z, font= ('Verdana',16))

window = tkinter.Tk() #create tkinter window
window.title("Addition") #give title
window.configure(background="white") #change background color
button_on = tkinter.Button(window, text="Add",
                        font= ('Verdana',16), padx=60, pady =5,
                        bg="green",fg="white",
                        command = add)
```

Function to perform addition

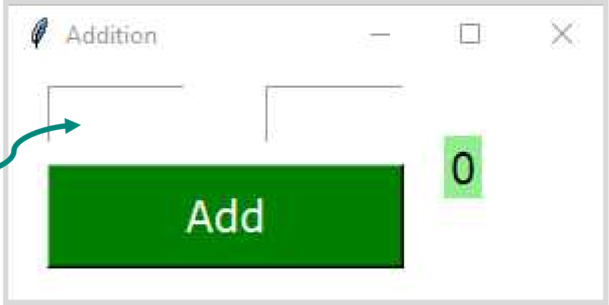
Button definition with command traceback

**Python**

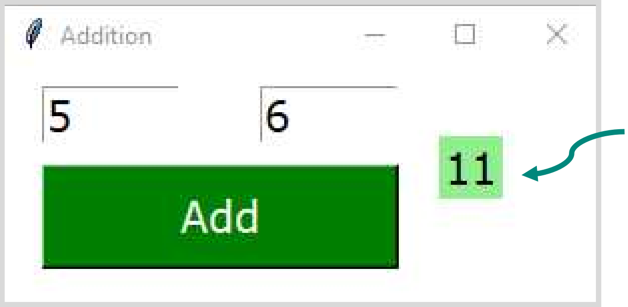
```
button_on.grid(row=1,column=0, columnspan=2)

in1 = tkinter.Entry(window, width = 5, borderwidth=1, font= ('Verdana',16))
in1.grid(row = 0, column =0, padx=20, pady =10)

in2 = tkinter.Entry(window, width = 5, borderwidth=1, font= ('Verdana',16))
in2.grid(row = 0, column =1, padx=20, pady =10)

label = tkinter.Label(window, text="0", font= ('Verdana',16), bg="lightgreen")
label.grid(row=0, column=2, rowspan=2)

window.mainloop()
```

Input fields

Output of added numbers



Enter numbers here



Output value

**What else can you do based on this example?**

**Try creating a simple calculator to perform basic arithmetic functions.**

## 3.7. DEMYSTIFYING PYSERIAL

Serial communication is a method or a process in which data is transmitted between devices/ systems at a sequential manner. It uses various communication channel to send/receive data between devices/ systems.

We know that our embedded electronics are made of multiple electronics components. And all these electronic components are interlinked to forma symbiotic system. Such complex systems needs a communication protocol and channel to talk between themselves and to external systems. One such method of sending data across systems is a serial communication.

This chapter assumes you have a basic knowledge in Serial and parallel communication protocol and how its being used in electronics. Since we will be dealing with Arduino based micro controllers, We need Serial data to be transmitted from or to Arduino at a specific baud rate. A baud rate is the speed at which data bits are transmitted over serial communication channel. In our case in following exercises, we will be transmitting data via USB serial cable from arduino to our Python program via USB port of our PC.

### PySerial

So, in order to communicate to external devices from Python, we need to make use of python modules built for this purpose. ***PySerial package*** helps  us to make such communication with external applications/ devices.

**Installation:**

1. Open Anaconda command prompt
2. Enter: ***`pip install pyserial`***
3. Complete the installation

Now that we had an overview of Tkinter widgets and programming them appropriately, let’s learn a bit on High level tkinter widgets available for our usage.

# 4 Explore High level widgets with TK-Tools

**This chapter gives you a glimpse of high level widgets that you can make use of in your GUI development. You will learn to import them into tkinter window, control them based on function calls and even get inputs from them for further use.**

## 4.1. HIGH LEVEL WIDGETS WITH TKTOOLS

The tk_tools module has advanced GUI implementations for Tkinter interface. The standard collection of widgets in Tkinter allows for limited configuration changes like colors, looks and basic functionalities. In order to enhance the process of building complex widgets, tk_tools allows for maximum customization of UI elements. We will setup the  tk_tools and use few widgets to explore it's capabilities.

**Installation:**

1. Open Anaconda command prompt
2. Enter: ***pip install tk-tools***
3. Complete the installation

**Python**

```
import tkinter
import tk_tools

window = tkinter.Tk()


# Place your Tkinter widgets here


window.mainloop()
```

Following widgets and widget categories are available in tk_tools module. Let's try to make use of one widget from each categories in our examples that follows.

**Widget Groups**

- LabelGrid
- EntryGrid
- ButtonGrid
- KeyValueEntry
- Calendar
- MultiSlotFrame
- SevenSegment

**Canvas Widgets**

- RotaryScale
- Gauge
- Graph
- LED

**Smart Widgets**

- SmartOptionMenu
- SmartSpinBox
- SmartCheckbutton
- SmartListBox
- BinaryLabel
- ByteLabel

Visit this GitHub to learn more about all the widgets mentioned above:

***https://github.com/slightlynybbled/tk_tools***

## 4.2. SEVEN SEGMENT DISPLAY

In this example, we will see how to use seven segment display widget in our tkinter window. We will take our previous example of addition of numbers. Instead of displaying our results in a text widget, we will display it in a seven segment display widget.

Python

```
import tkinter
import tk_tools
```

Function of adding 2 numbers

```
def add():
    x = int(in1.get())  #get value from entry box
    y = int(in2.get())  #get value from entry box
    z = x + y
    #Convert the integer value to string to display in Seven segment widget
    ss_int.set_value(str(z))  # Update the value of seven segment display widget
```

Initialize Tkinter widget

```
window = tkinter.Tk() #create tkinter window
window.title("Addition") #give title
window.configure(background="white") #change background color
window.geometry("400x110")
```

Button to perform addition function

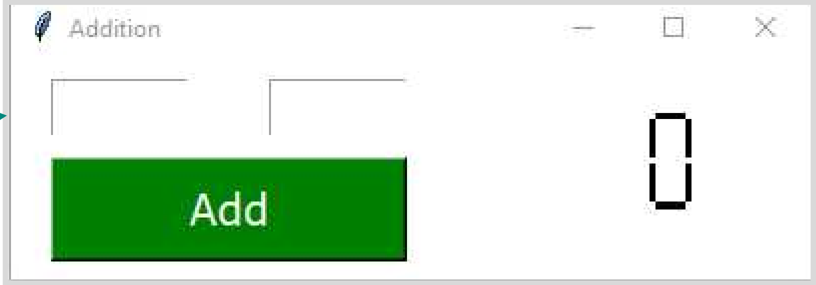
```
button_on = tkinter.Button(window, text="Add",
                           font= ('Verdana',16), padx=60, pady =5,
                           bg="green",fg="white",
                           command = add)
button_on.grid(row=1,column=0, columnspan=2)
button_on.grid(row=1,column=0, columnspan=2)
```

Declare Entry widgets

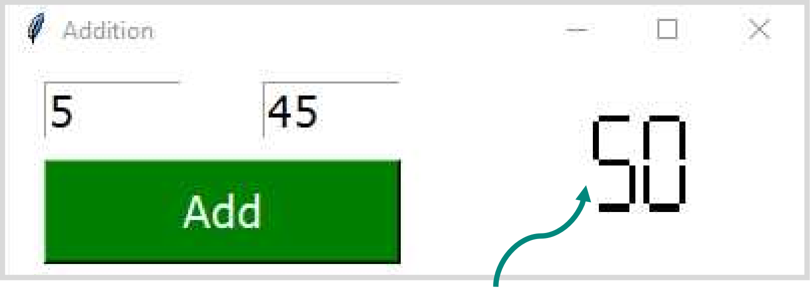
```
in1 = tkinter.Entry(window, width = 5, borderwidth=1, font= ('Verdana',16))
in1.grid(row = 0, column =0, padx=20, pady =10)

in2 = tkinter.Entry(window, width = 5, borderwidth=1, font= ('Verdana',16))
in2.grid(row = 0, column =1, padx=20, pady =10)
```

Seven segment widget definition

```
ss_int = tk_tools.SevenSegmentDigits(window, digits=5, background='white', 
digit_color='black')
ss_int.grid(row=0, column=6, rowspan=2)
ss_int.set_value(str(0))
window.mainloop()
```



Input your values here



5

45

Seven segment display

## 4.3. ROTARY SCALE

Next, we will try to implement one of the ***canvas widgets category***. These widgets provides users with visual output. Even though you can create such widgets on your own (which will take you time to build), its good to start with ready to use  widgets.

This time, we will make it interesting. Lets try to use ***rotary scale widget*** and control it according to slider position of our ***scale widget***.

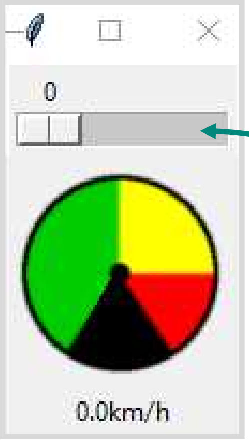
**Python**

```
import tkinter
import tk_tools


def update(y):
    rs.set_value(int(y))
window = tkinter.Tk() #create tkinter window
window.title("Speed control") #give title
window.configure(background="white") #change background color
window.geometry("120x180")
scale = tkinter.Scale(window, from_=0, to=100, orient=tkinter.HORIZONTAL,
command = update)
scale.grid(row=0, column=0)
rs = tk_tools.RotaryScale(window, max_value=100.0, size=100, needle_thickness=5, needle_color='black',unit='km/h')
rs.grid(row=1, column=0)
window.mainloop()
```

Function to update Rotary scale widget

Slider bar widget

Rotary scale widget definition



Output

Move the slider to control the rotary scale

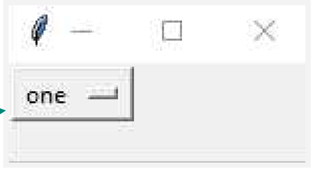
## 4.4. SMART OPTIONS MENU

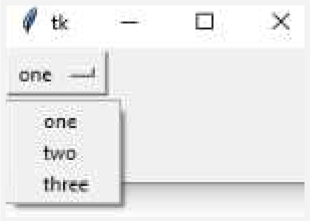
In this last example, we will see how to use smart options menu widget in our tkinter window. This widget provides a Classic drop down entry with built-in tracing variable compared to tkinter menu widget.

**Python**

```
import tkinter
import tk_tools

window = tkinter.Tk()
# create the dropdown and grid
som = tk_tools.SmartOptionMenu(window, ['one', 'two', 'three'])
som.grid()

# define a callback function that retrieves
# the currently selected option
def callback():
    print(som.get())

# add the callback function to the dropdown
som.add_callback(callback)
window.mainloop()
```

Smart Option widget creation



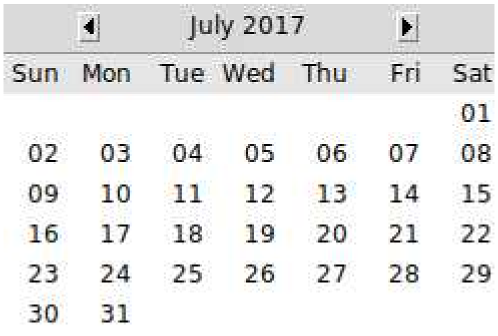


Menu button

## OTHER WIDGETS TO TRY

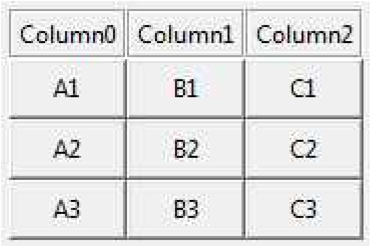
There are other widgets to try and implement in your Tkinter Project depending upon the need. Some examples could be the use of Entry widgets, Calendar widgets and so on. It’s a best approach to look into documentation page and understand how other widgets work in its functionality and implementation. Some examples are shown below.

Calendar widget



◀ July 2017 ▶

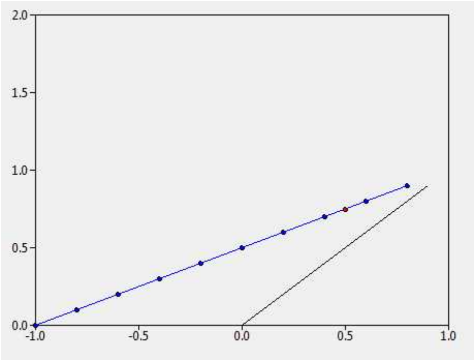
| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| | | | | | | 01 |
| 02 | 03 | 04 | 05 | 06 | 07 | 08 |
| 09 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | | | | | |



| Column0 | Column1 | Column2 |
|---|---|---|
| A1 | B1 | C1 |
| A2 | B2 | C2 |
| A3 | B3 | C3 |

Button Grid

Graph widget



# EXERCISES

**USEFUL TIPS FOR TROUBLESHOOTING ERRORS:**

**Arduino:**

- Every time before you Upload a new Sketch to arduino board, Make sure to choose the correct COM Port!
- Close the Arduino's Serial Monitor while communicating to Arduino microcontroller from Python IDE

**Python:**

- If you get "*Serial Port busy error*" or "Could not open port 'COM6'' error while communicating with Arduino, Unplug the Arduino cable and plug it back in and try again. Also make sure to use correct COM port of your Arduino is connected with in your Python program.
- If you get "*PyImage doesn't exist*" error, Please restart your Kernel / Close your Spyder IDE and reopen again.

# 5

## Challenge-1

# DIGITAL CONTROL OF LED

**Control an LED using an Arduino micro controller with a help of script written in Python language. You will be creating a GUI consisting of a tkinter button with an interactive LED interface to control the LED connected to your Arduino. We will make use of Serial module from python to send or receive data from Arduino to our python GUI.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, tkinter-tools to control the state of an LED connected to a micro-controller. You will learn to create buttons, arrange buttons on tkinter window, assign a state of a button based on the event call. You will also learn to create variables & define function calls.

Refer to GitHub repository mentioned in Page 139 for the codes:

***ex1_led_control.py***

***e1_led_control.ino***

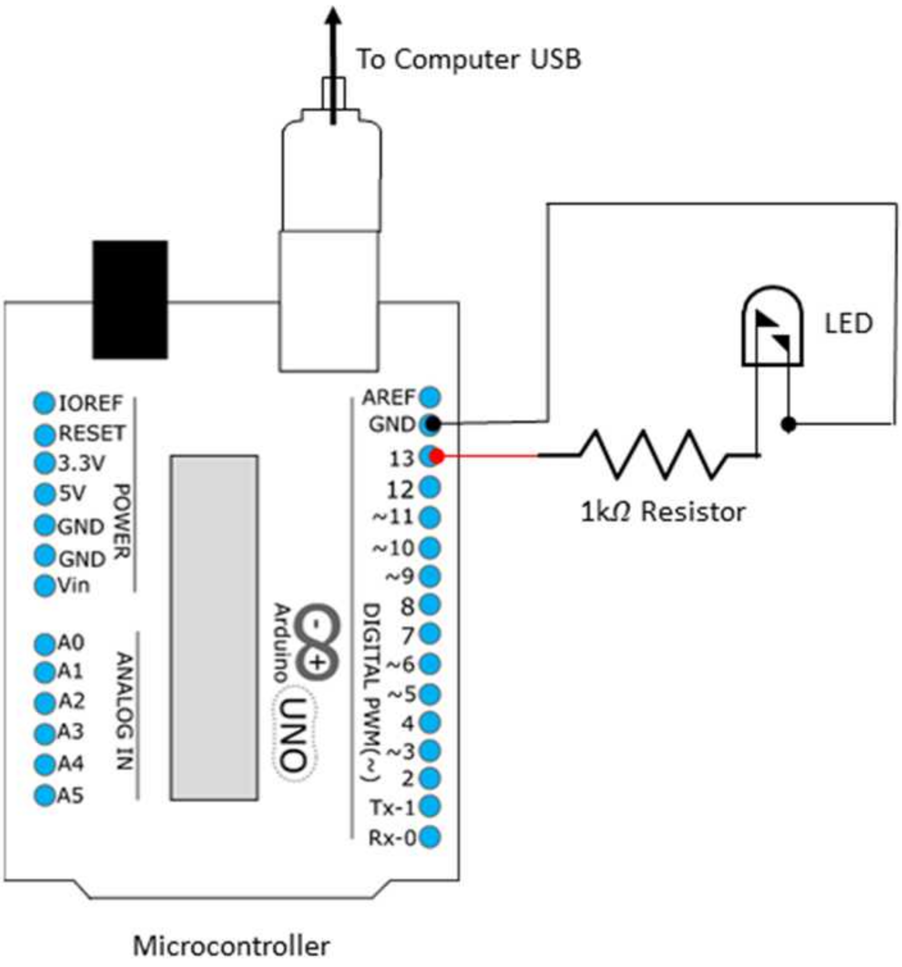
## Exercise

This exercise is about controlling a LED connected to arduino micro-controller from a computer using a GUI built with Python script.

## Hardware required

1. Arduino Uno – 1 Pc
2. LED – 1 Pc
3. Jumper wires (M to F)- 2 Pc
4. Bread Board—1 Pc (optional)

## Circuit schematic



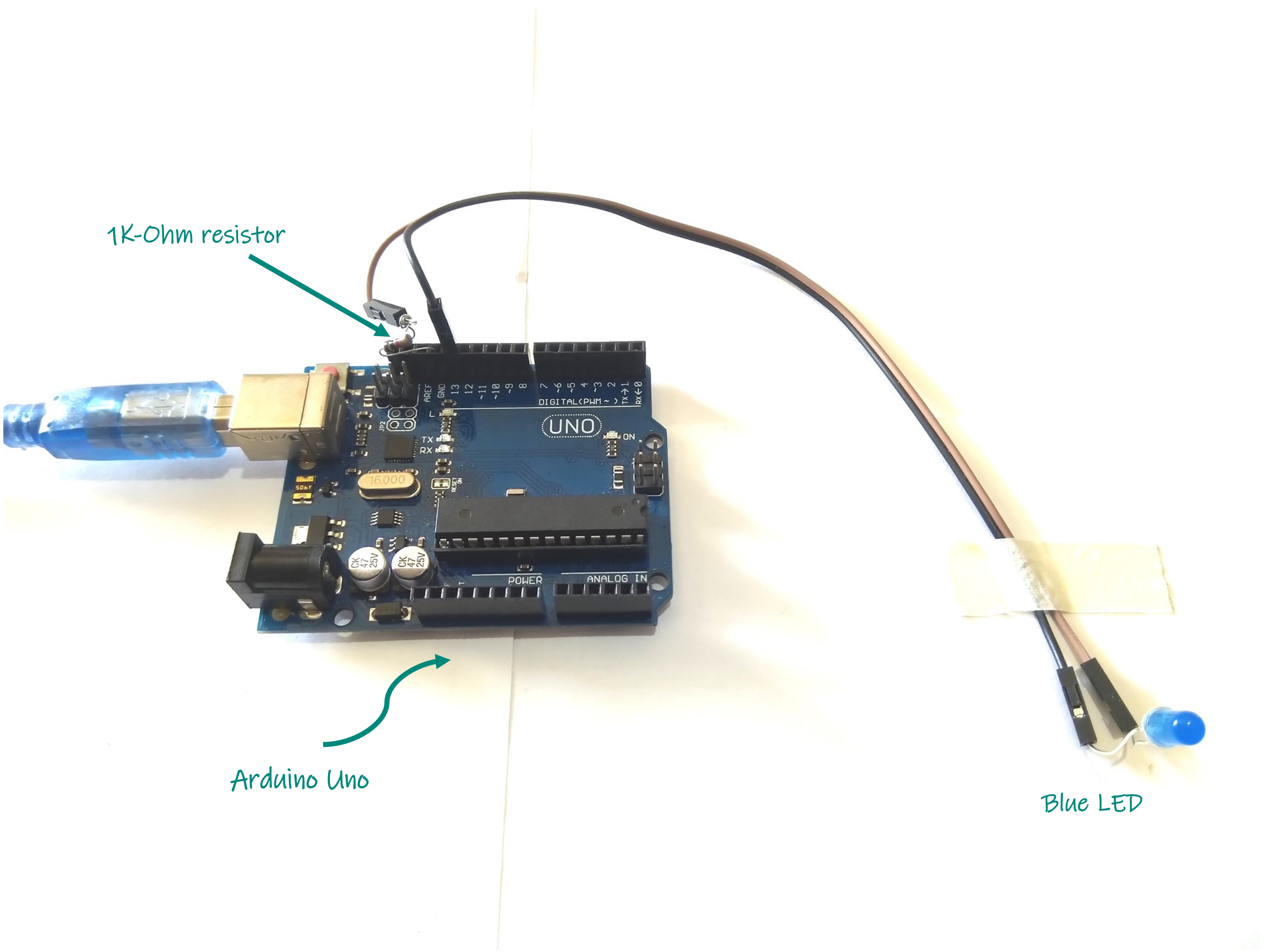
You can also prefer to use inbuilt Arduino's LED at PIN 13

**Circuit connections explainer:**

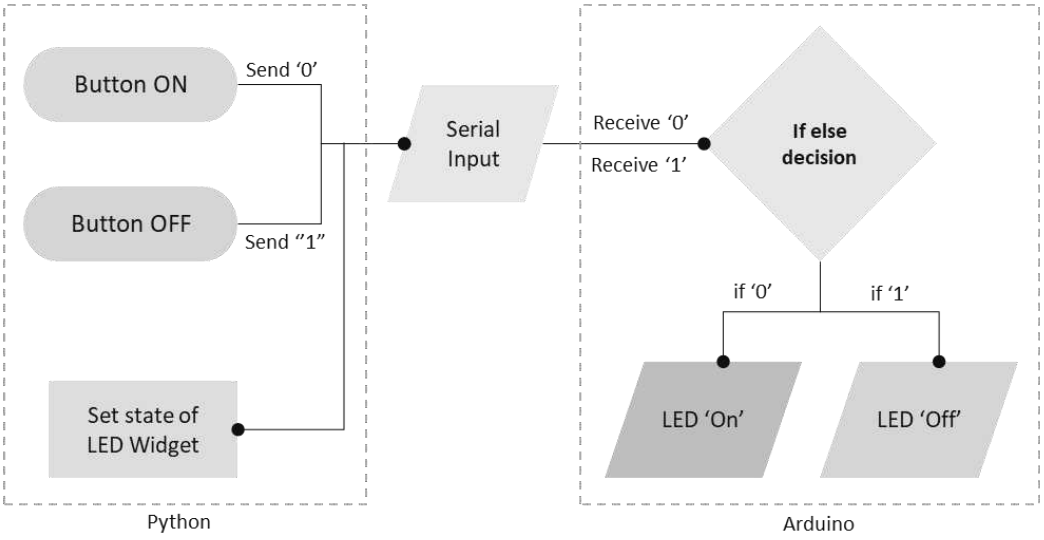
LED's anode (longer leg) is connected to PIN 13 of arduino

LED's cathode (shorter leg) is connected to GND of arduino via a resistor

# Hardware setup



## Work Flow



## Arduino script

```
//Declare variable
char  serialData;

//Setup your baudrate & arduino pins
void setup()
{
   pinMode(13, OUTPUT);      // LED pin on arduino
   Serial.begin(9600);
}

//Run your program again and again using loop
void loop()
{
 if (Serial.available() > 0) // send data only when you receive data from python
 {
     serialData = Serial.read(); //read serial data and assign to variable
     Serial.print(serialData);   //Print to check your data
     //Condition statement
     if(serialData == '0')
     {
         digitalWrite(13, HIGH); //Make your LED glow
      }
     }
else if(serialData == '1')
    {
        digitalWrite(13, LOW); //Make your LED off
    }
 }
```

***How it works***

This arduino sketch assigns the pin 13 to be an output. This is where your LED is connected with. You can also make use of arduino's inbuilt LED to perform this operation. At the beginning of the loop, a serial read is triggered to read any data that comes in via serial port. Once the data is read, an ***if_else*** condition is set to change the LED's state. If the incoming data is '0', LED is set to HIGH and if incoming data is '1', the LED is set to LOW state.

***Verification***

Once you compile and upload this sketch to your arduino, then try to send '0' or '1' via arduino IDE's ***serial monitor***. This should make your LED to turn ON/OFF accordingly. This is a basic sketch to begin with our exercises. Once you understand the logic of if_else statements to trigger an event, then its convenient to make other complex functions. You will get to explore this in upcoming exercises.

## Python script break down

***Import Packages:*** Lets begin with Python script to build our GUI. At first we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tk-tools: a** package for creating ready made interactive GUI.

```
import serial
import tkinter
import tk_tools
```

***Establish Serial Connection:*** Next we need to establish a serial connection to our controller. The method "*serial.Serial('COM6', 9600)*" establishes a serial connection to COM port 6 at a baud rate of 9600.

```
arduino = serial.Serial('com6', 9600)
```

***Create tkinter window:*** A Tkinter window can be created using tkinter.TK method. You can add titles, fix geometry, change background color of your window and so on.

```
window = tkinter.Tk()                          #create tkinter window
window.title("Arduino RGB Led Control")        #give title
window.configure(background="white")           #change background color
```

***Create buttons:*** Based on our exercise, we are going to create 3 buttons for our GUI. A button to switch ON the led, a button to switch OFF the led and a button to destroy tkinter window and close our GUI. Each button is linked with its own function call and configurations. You can assign the size, position using a grid method, text size, text type, foreground color, state of button and so on. For this exercise we will use the following arguments for our buttons. Following is an example of button_on that is created in tkinter window. Refer to figure (a) for the display of our button.

```
button_on = tkinter.Button(window, text="ON",
                    font= ('Verdana',16), padx=50, pady =20,
                    bg="green",fg="white",
                    command = led_on)
```

***Pack buttons:*** Once we create the buttons according to our need, then we have to pack them in our tkinter window. There are methods such as Pack, Grid and place are available. We will use grid method to pack our buttons according to desired columns and rows.

```
button_on.grid(row=1,column=1)
button_off.grid(row=1,column=2)
```

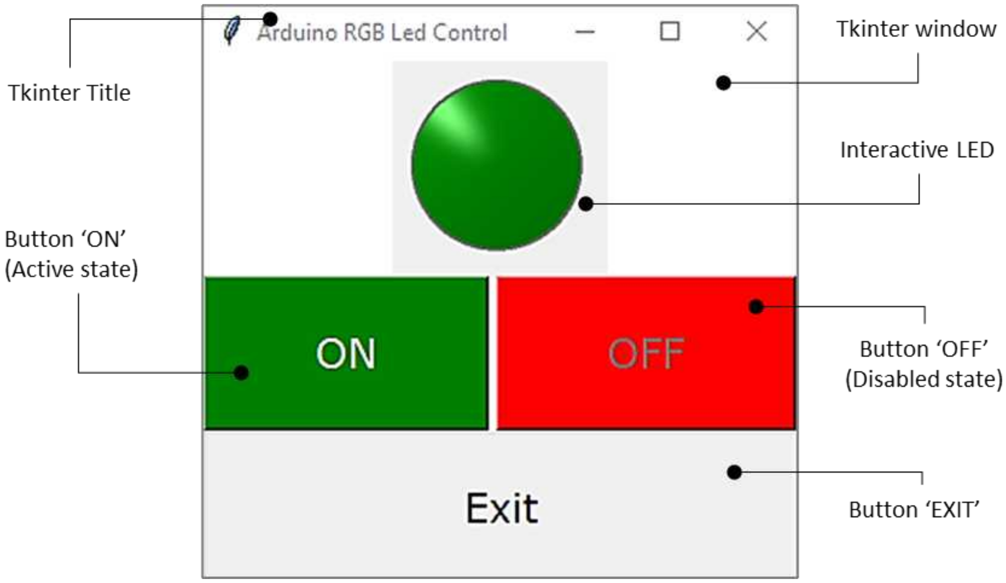
***Function call:*** We have to define a function call to perform events whenever a button is pressed/Clicked. In our case, if button *ON* is clicked, python writes a value ‘0’ to arduino via serial Write method. After this, we can set the button to turn into *disabled* state using *widget config* method of tkinter. This will add an intuitiveness to our GUI. Since we also have an interactive LED (from tk-tools module) in our tkinter window, we can also set it to glow state by giving a *True* value to led.to_green() function call.

```
def led_on():
    arduino.write(b'0')                          #write this value to arduino
    led.to_green(True)                           #Glow the led to green
    button_on.config(state = "disabled")         #disable button_on
    button_off.config(state = "normal")          # enable to normal state
```

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list.

- *Connect your hardware according to circuit diagram provided.*
- *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Verify the program using Arduino IDE Serial monitor.*
- *Once verified, Close the Serial monitor of Arduino IDE.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! You will see your GUI opened.*

***Tkinter GUI***

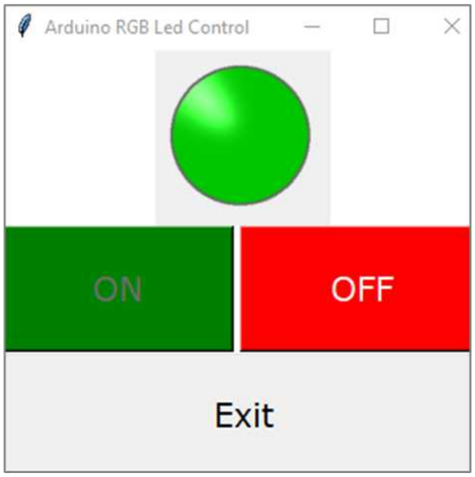


## Python script

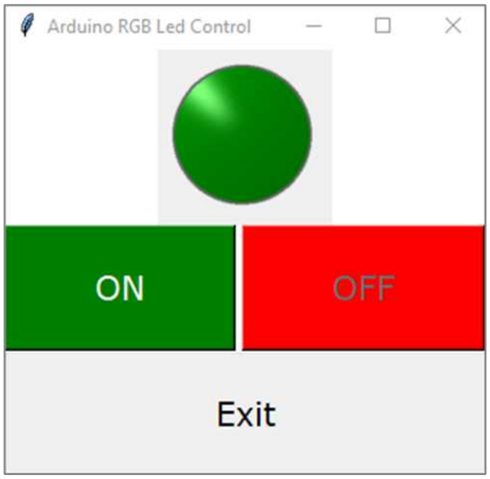
```
#Import following packages
import serial
import tkinter
import tk_tools

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial('com6', 9600)

#Function call for led_on button
def led_on():
    arduino.write(b'0') #write this value to arduino
    led.to_green(True)  #Glow the led to green --> Refer to tk-tools docs
    button_on.config(state = "disabled") #disable button_on
    button_off.config(state = "normal") # enable button_off to normal state

#function call for led_off button
def led_off():
    arduino.write(b'1') #write this value to arduino
    led.to_green() #Switch off the led --> Refer to tk-tools docs
    button_off.config(state = "disabled") #disable button_off
    button_on.config(state = "normal") #enable button_on to normal state

#exit function call for closing the program
def close_window():
    arduino.close() #close Serial connection with arduino
    window.destroy() #destro tkinter app

# MAIN
window = tkinter.Tk() #create tkinter window
window.title("Arduino RGB Led Control") #give title
window.configure(background="white") #change background color

#Create buttons
button_on = tkinter.Button(window, text="ON",
                           font= ('Verdana',16), padx=50, pady =20,
                           bg="green",fg="white",
                           command = led_on)
```

```
button_off = tkinter.Button(window, text="OFF",
                            font=('Verdana',16), padx=50, pady =20,
                            bg="red",fg="white",
                            command = led_off)

button_exit = tkinter.Button(window, text="Exit",
                             font=('Verdana',16),
                             padx=130, pady =20,
                             command = close_window)

#Pack buttons
button_on.grid(row=1,column=1)
button_off.grid(row=1,column=2)
button_off.config(state = "disabled")
button_exit.grid(row=2,column=1, columnspan =2)

#Create interactive led using tk-tools
led = tk_tools.Led(window, size=100)
led.to_green() #set led to green switch off condition) --> Refer tk-tools docs
led.grid(row=0,column=1, columnspan =2)

#execute the loop
window.mainloop()
```

***Python GUI Output***



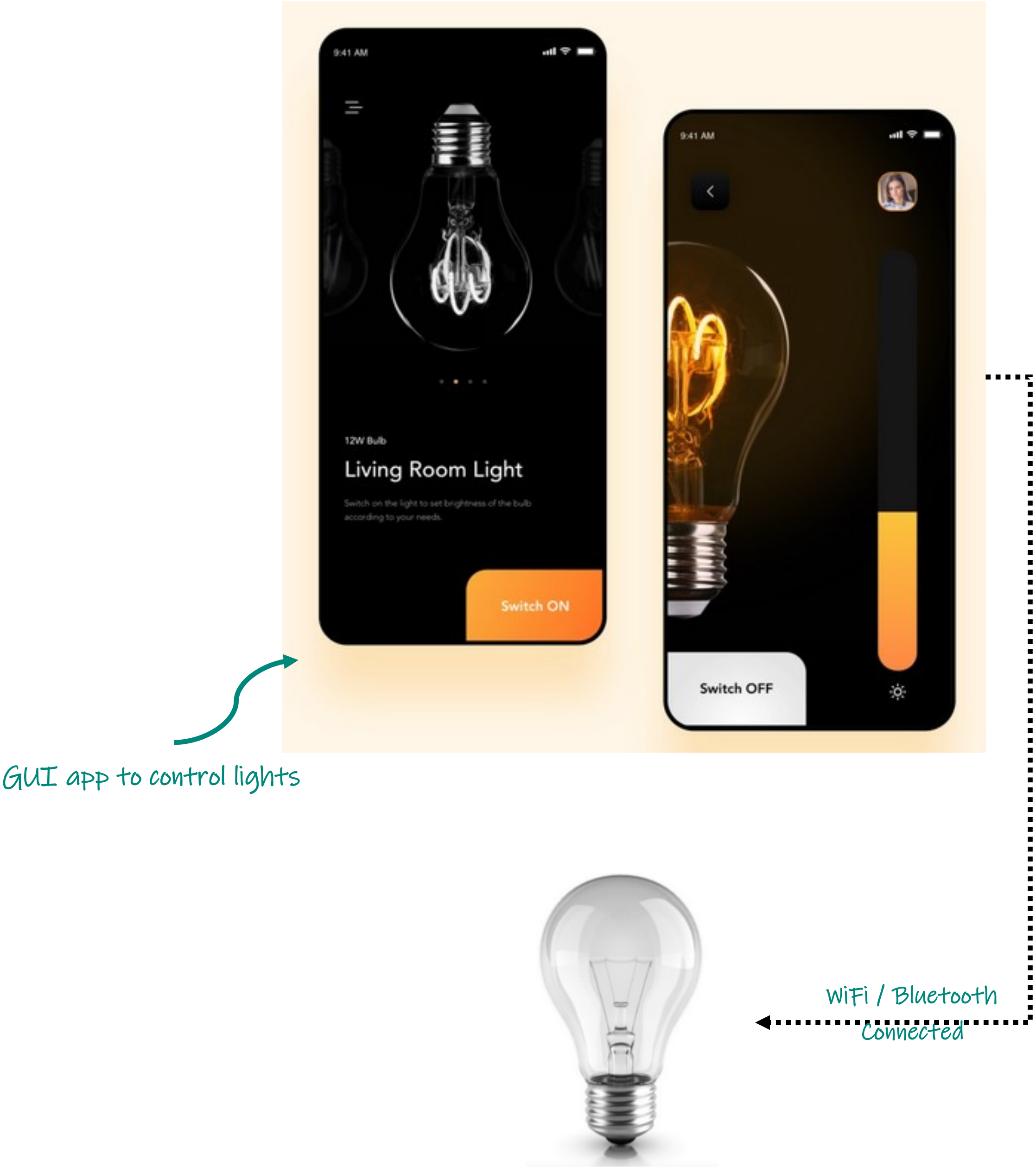
LED 'ON' State



LED 'OFF' State

## POTENTIAL APPLICATIONS

- Automation of Industrial process flow with Switches and light indicators with GUI
- To output the status of Smart lights in our homes with an interactive mobile GUI.


9:41 AM
12W Bulb
Living Room Light
Switch on the light to set brightness of the bulb according to your needs.
Switch ON
9:41 AM
Switch OFF
GUI app to control lights
WiFi / Bluetooth
Connected

## CONCLUSION

In this chapter, we understood the use of Tkinter button widget and created a program to control the state of LED’s.

# 6

## Challenge-2

# CONTROL OF DIGITAL SERVOS

**Control a digital servo motor using an Arduino micro controller with a help of script written in Python language. You will be creating a GUI consisting of a tkinter slider widget with an interactive dial gauge interface to set a position angle to a digital servo motor connected to your Arduino. We will make use of Serial module from python to send or receive data from Arduino to our python GUI.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, tkinter-tools to control the position of a servo motor connected to a micro-controller. You will learn to create slider widgets, dial gauge widget, arrange buttons on tkinter window, send a angle data to micro-controller based on the slider position. You will also learn to create variables & define function calls.

Refer to GitHub repository mentioned in Page 139 for the codes:

***e2x_servo_control.py***

***e2_servo_control.ino***

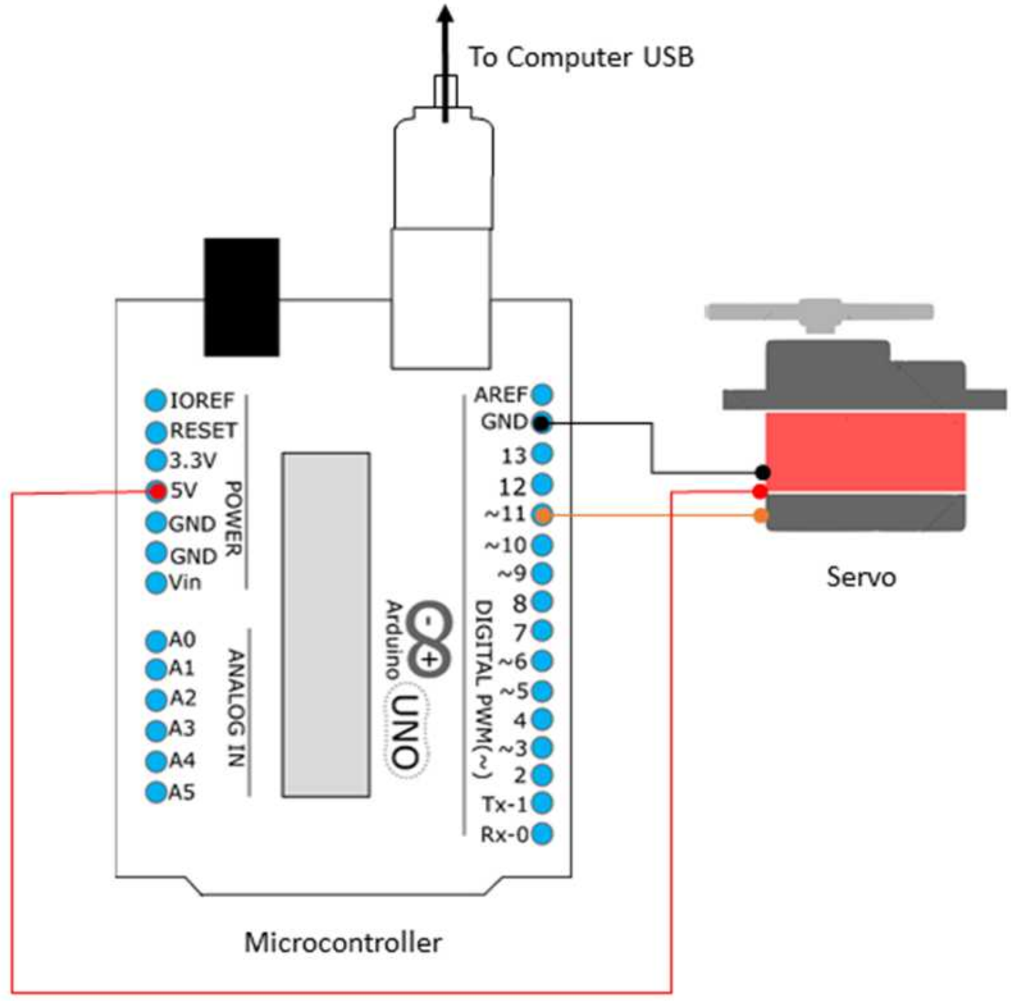
## Exercise

This exercise is about controlling a Servo motor connected to arduino micro-controller from a computer using a GUI built with Python script.

## Hardware required

1. Arduino Uno – 1 Pc
2. Digital servo motor – 1 Pc
3. Jumper wires (M to M)- 3 Pc
4. Bread Board—1 Pc (optional)

## Circuit schematic



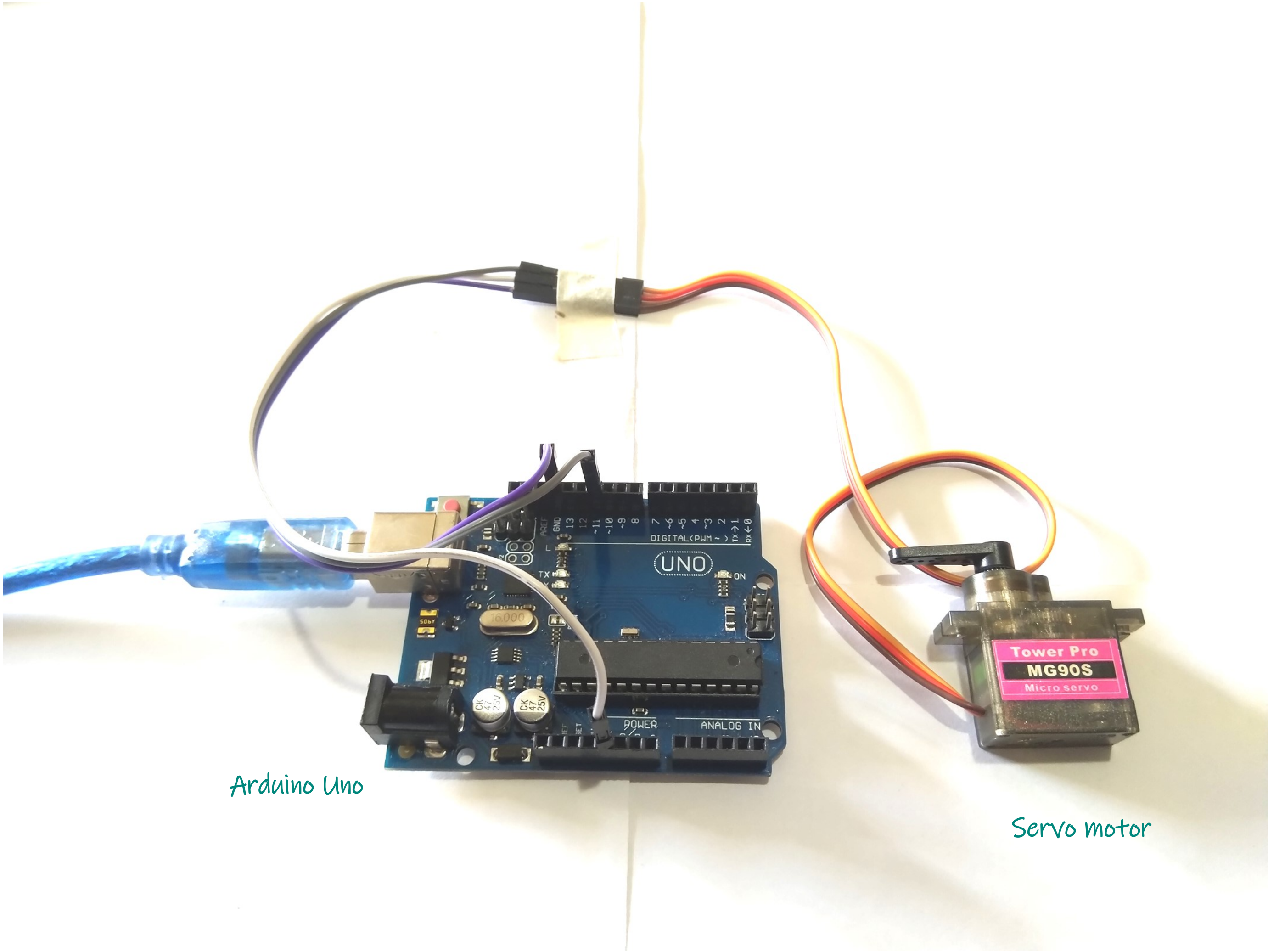
**Circuit connections explainer:**

Servo motor GND wire( Brown/Black) is connected to GND PIN of arduino

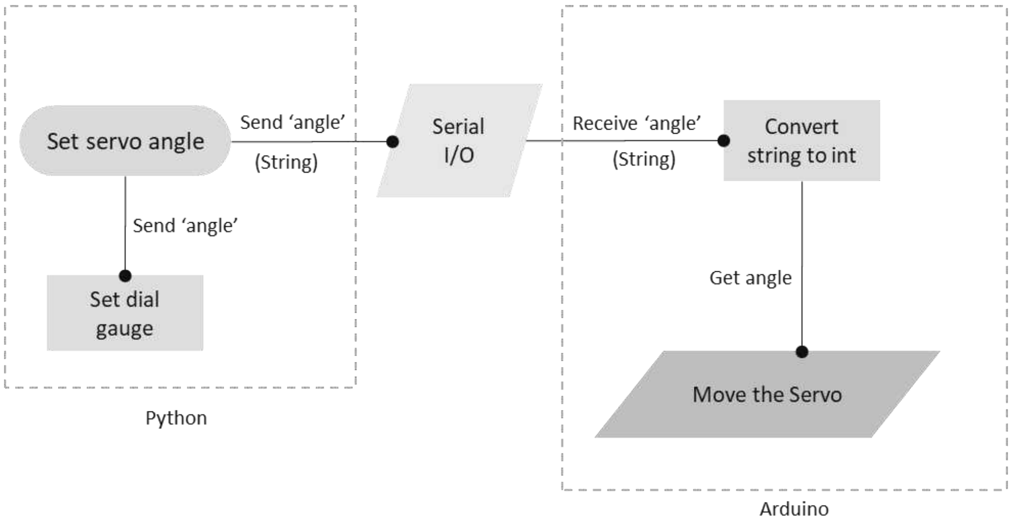
Servo moto VIN wire (Red) is connected to 5V PIN of arduino

Servo motor SIG wire (Yellow/Orange/White) is connected to PIN 11 of arduino

# Hardware setup



## Work Flow



## Arduino script

```
#include <Servo.h> //Include servo library
Servo myservo;
//Declare variables
String inByte;
int pos;

void setup() {
  myservo.attach(11);     //Digital pin that your servo is attached
  Serial.setTimeout(80);  //time in which Serial port will wait for serail data
  Serial.begin(9600);
}

void loop(){
        // send data only when you receive data from python
        while (Serial.available() > 0)
        {
                // read the incoming byte:
                String c = Serial.readString();
                //Convert string to integer
                pos = c.toInt();
                delay(2);
```

```
            // Print your value back
            Serial.print("C is ");
            myservo.write(pos);
            Serial.println(pos);
      }
}
```

***How it works***

This arduino sketch assigns the pin 11 to be an output. This is where your servo signal wire is connected with. You can also connect to other PWM pins of arduino to perform this operation. At the beginning of the loop, a serial read is triggered to read any data that comes in via serial port. The incoming-data is read as a string value, it is then converted to in type using ***string.toInt()*** function. Then this int value is sent to ***servo.write()*** function to set the position of servo.

***Verification***

Once you compile and upload this sketch to your arduino, then try to send values from 0 to 180 via arduino IDE's ***serial monitor***. This should make your servo to move to respective angle accordingly as well as it prints the reached angle into your serial monitor. If it works, then you are good to go with Python scripts for creating your GUI.

## Python script break down

Lets begin with Python script to build our GUI. As referred with exercise 1, we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tk-tools: a** package for creating ready made interactive GUI. Then establish a serial connection by assigning a respective COM ports. As described in previous exercise, create a tkinter window.

***Scale widget:*** Next, instead of creating buttons, we will be creating a ***Scale*** widget. You can set the dimensions of your scale, limits of your scale, (in our case its 0 to 180) and assign a event call using command method as shown below. Once you move your slider in scale widget, the respective values can be pulled stored in a variable and can used further.

```
servo = tkinter.Scale(window, activebackground="blue",
                label = "        Set the Angle of your servo",
                bg = "white", font=('Verdana',16), from_=0, to=180,
                orient=tkinter.HORIZONTAL, length= 400,
                command = move_servo)
```

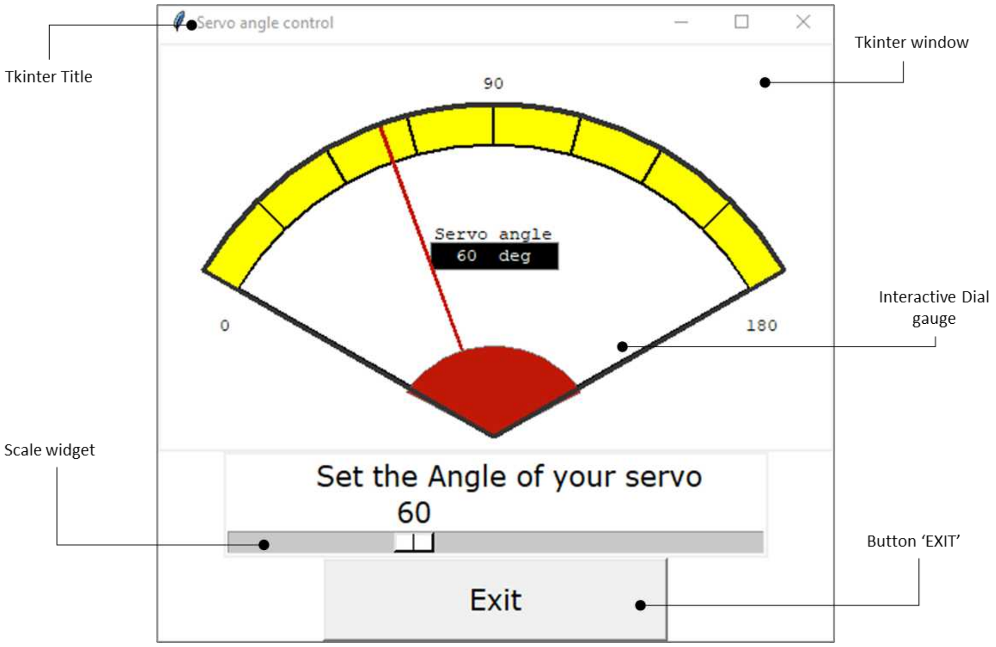
***Dial Gauge widget:*** In this GUI, we will be using an interactive dial gauge widget for visualizing our angle. The dial in the widget will move to a respective position from 0 to 180 degrees based on the user's slider position input by using the function *speed_gauge.set_value(var).*

```
speed_gauge = tk_tools.Gauge(window, max_value=180, bg='white',
                          label='Servo angle', unit='  deg',
                          red=90, yellow=10, height = 300, width = 500)
```

Then we pack the widgets in tkinter window using grid method and place them in respective rows and columns.

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- *Connect your hardware according to circuit diagram provided.*
- *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Verify the program using Arduino IDE Serial monitor.*
- *Once verified, Close the Serial monitor of Arduino IDE.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! You will see your GUI opened.*

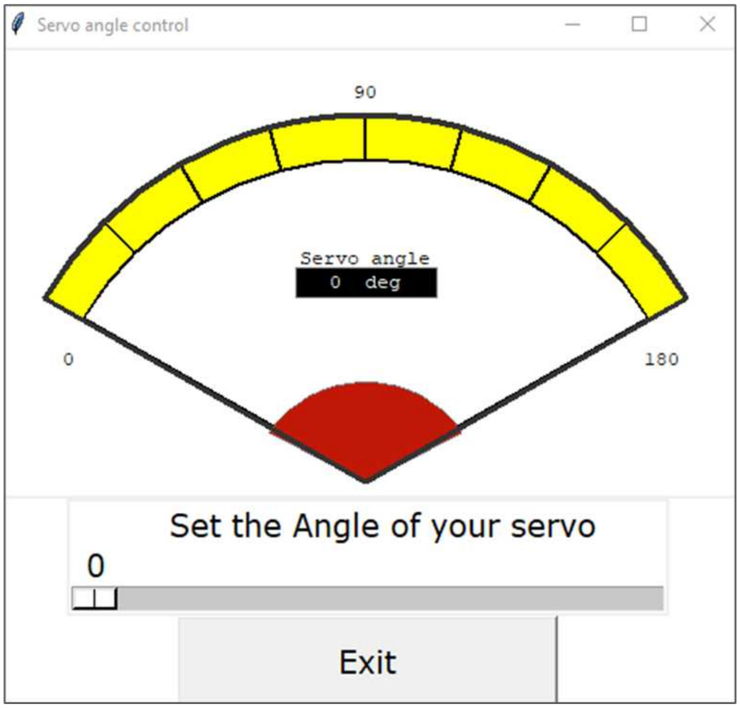


***Tkinter GUI***

## Python script

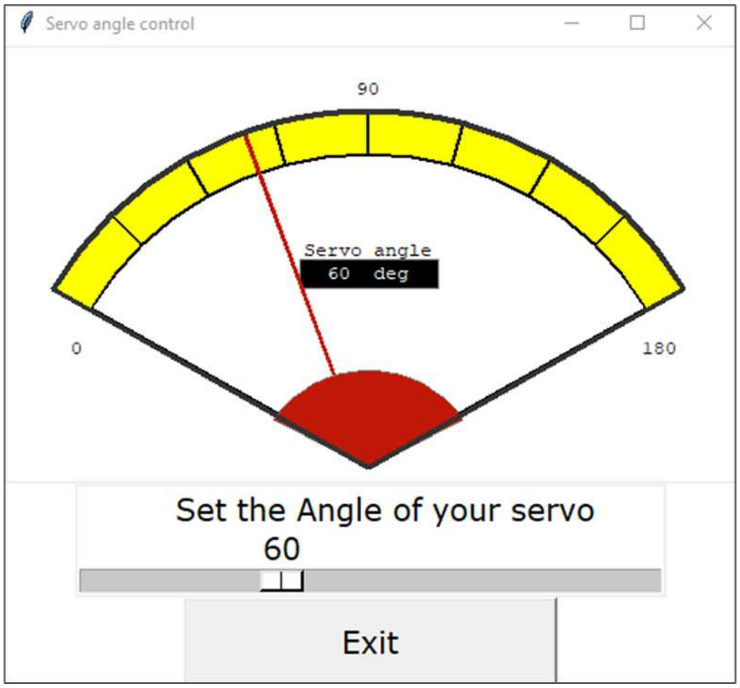
```
#Import following packages
import serial
import tkinter
import tk_tools

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial(‘com6',9600)

#Function call to close your program
def close_window():
   arduino.close()
   window.destroy()

#Function call to move_servo button
def move_servo(var):
   speed_gauge.set_value(var) #set value to the speed gauge
   print(var) #print value for cross-check

   #send servo angle value to arduino
   arduino.write(str(var).encode()) #write this value to arduino

   #get servo angle  ack from arduino to cross-check
   reachedPos = str(arduino.readline()) #get value back from arduino
   print (reachedPos)

# MAIN
window = tkinter.Tk() #create tkinter window
window.title("Servo angle control")  #give title
window.configure(background="white") #change background color

# Create a slider for servo position
servo = tkinter.Scale(window, activebackground="blue",
                label = "        Set the Angle of your servo",
                bg = "white", font=('Verdana',16), from_=0, to=180,
                orient=tkinter.HORIZONTAL, length= 400,
                command = move_servo)
```

```
#Create a speed-gauge using tk-tools module
speed_gauge = tk_tools.Gauge(window, max_value=180, bg='white',
                 label='Servo angle', unit='  deg',
                 red=90, yellow=10, height = 300, width = 500)

button_exit = tkinter.Button(window, text="Exit", font=('Verdana',16),
                 padx=100, pady = 10,
                 command=close_window)

#pack your buttons, sliders
servo.grid(row=1, column=0)
speed_gauge.grid(row=0, column=0)
button_exit.grid(row=2,column=0, columnspan =2)

#execute the loop
window.mainloop()
```



Inactive state

***Python GUI Output***

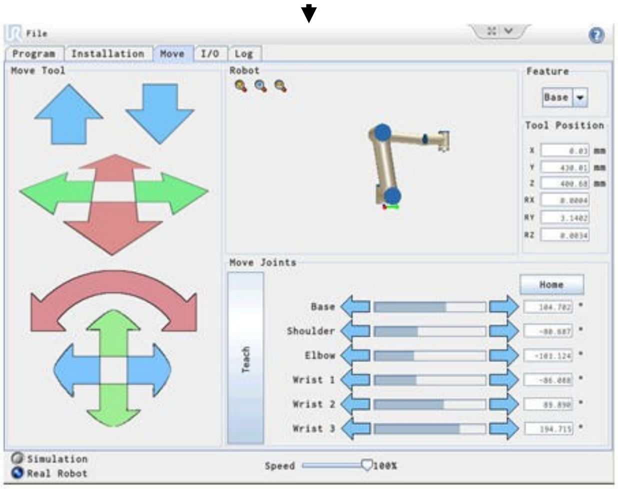


Active state

## POTENTIAL APPLICATIONS

- Control of opening and closing of Electric Window shades
- Controls of rudders/elevators in robot aircrafts and boats
- Joint control of robotic arms using a interactive GUI

Robotic arm



X 0.03 mm
Y 430.01 mm
Z 400.68 mm
RX 0.0004
RY 3.1402
RZ 0.0034

Base 104.702 °
Shoulder -80.687 °
Elbow -103.124 °
Wrist 1 -86.088 °
Wrist 2 89.890 °
Wrist 3 194.715 °

Speed 100%

A GUI for controlling the Joints of Robotic arm

## CONCLUSIONS

In this chapter, we controlled servo motors using a python program and Tkinter GUI.  This is useful in building servo based control arms and robot's.

# 7

## Challenge-3

# DC MOTOR CONTROL

**Control a DC (Direct current) Motor using an Arduino micro controller with a help of script written in Python language. You will be creating a GUI consisting of a tkinter slider widget with an interactive dial gauge interface to set a speed to a DC motor connected to your Arduino. We will also create direction control button to spin the motor in desired direction. We will make use of Serial module from python to send or receive data from Arduino to our python GUI.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, tkinter-tools to control the speed and direction of rotation of a dc motor connected to a micro-controller. You will learn to create slider widgets, dial gauge widget, arrange buttons on tkinter window, send a speed and direction data to micro-controller based on the slider position and button event calls.

Refer to GitHub repository mentioned in Page 139 for the codes:

***e3x_dc_motor_control.py***

***e3_dc_motor_control.ino***

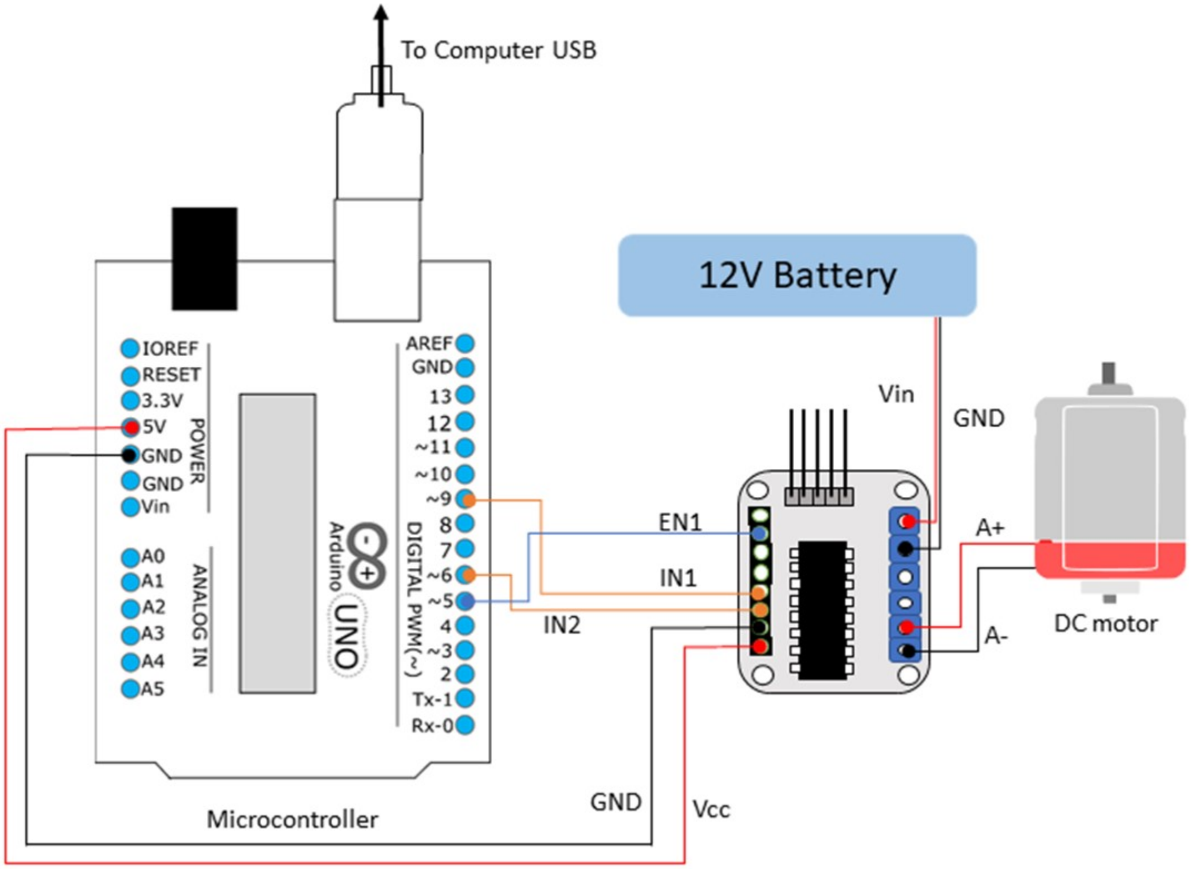
## Exercise

This exercise is about controlling a DC Motor connected to arduino microcontroller from a computer using a GUI built with Python script.

## Hardware required

1. Arduino Uno – 1 Pc
2. DC motor – 1 Pc
3. Jumper wires (M to M)- Few Pcs
4. Bread Board—1 Pc (optional)
5. Any L293D 2 channel motor driver of your choice—1 Pc
6. 9V or 7.4V Battery  of your choice—1 Pc

## Circuit schematic



**Circuit connections explainer:**

EN1 PIN of motor driver is connected to PIN 5 of arduino

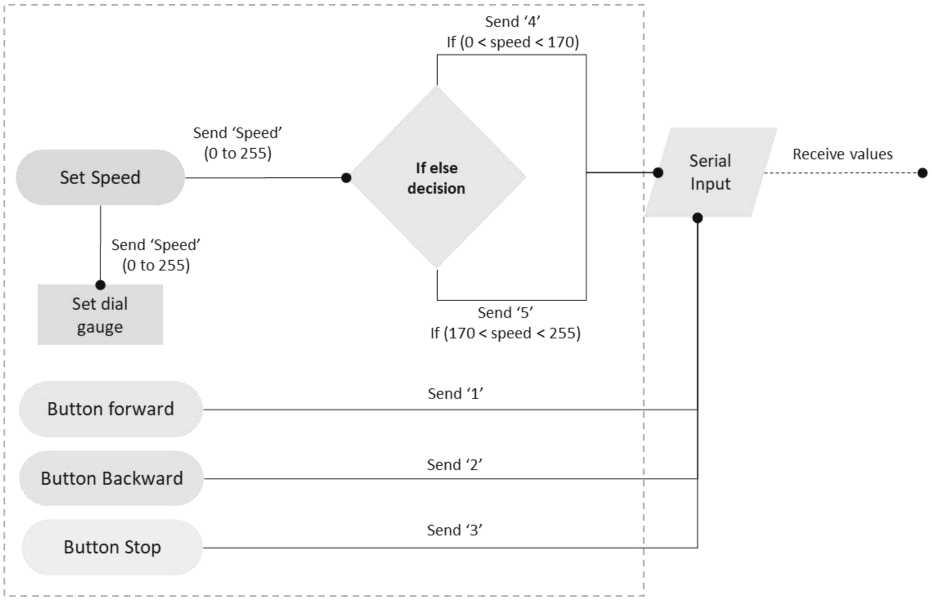
IN1 PIN of motor driver is connected to PIN 9 of arduino

IN2 PIN of motor driver is connected to PIN 6 of arduino

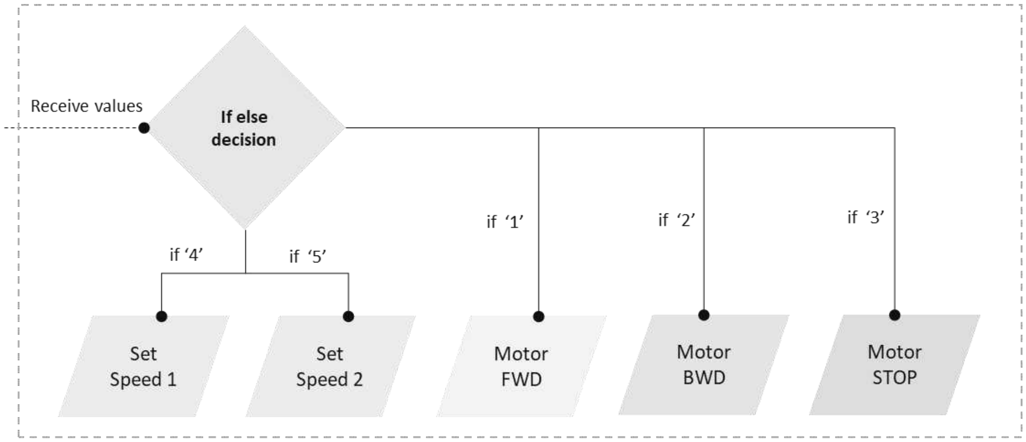
VCC PIN of motor driver is connected to 5V PIN of arduino

GND PIN of motor driver is connected to GND PIN of arduino

## Work Flow

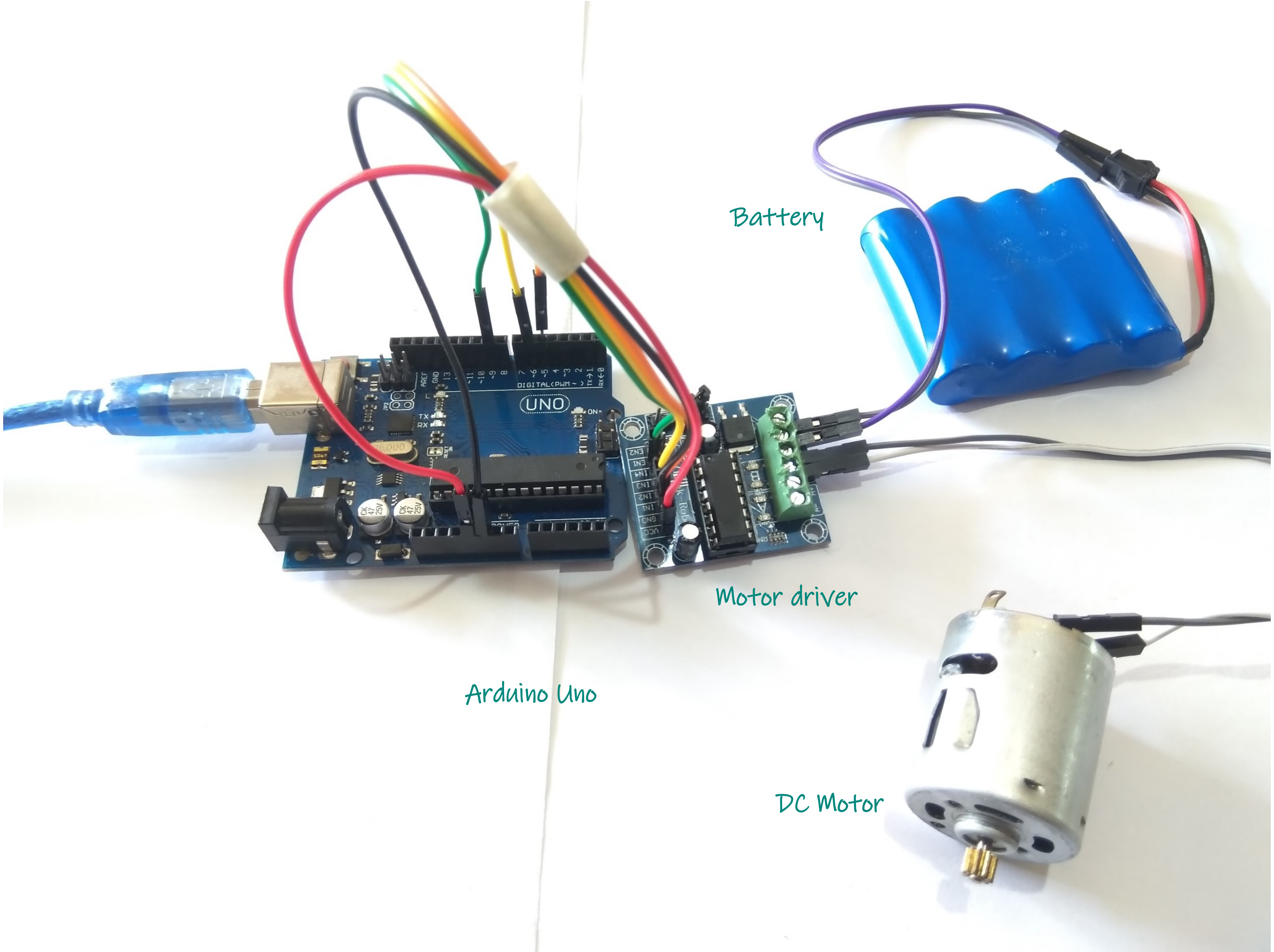


Python



Arduino

# Hardware setup



## Arduino script

```
//Declare variables
char  serialData;

//Declaring Pins of arduino to pins of Deek robot 2 Channel motor driver (L293D)
//Play around your driver to control the direction of motors
int m1=6; //connects to IN2 pin of Deek Robot motor driver -->Sets the direction
int m2=9; //Connects to IN1 pin of Deek Robot motor driver --> Sets the direction
int e1=5; //Connects to EN1 pin of Deek Robot motor driver -->Sets the speed

int x =0; //Variable to store the speed value from Python

void setup() {
      pinMode(m1, OUTPUT);
      pinMode(m2, OUTPUT);
      pinMode(e1, OUTPUT);
      Serial.begin(9600);
   }

void loop() {
   // send data only when you receive data from python
   if (Serial.available() > 0)
   {
     serialData = Serial.read();
     Serial.print(serialData);
     if(serialData == '1')  //Clock wise rotation of motor
     {
       analogWrite(e1, x); //Set the speed
       digitalWrite(m2, 0); //Clock wise rotation
       digitalWrite(m1, 1);
      }
      else if (serialData == '2')  //Anti-Clock wise rotation of motor
       {
        analogWrite(e1, x); //Set the speed
        digitalWrite(m2, 1); //Anti-Clock wise rotation
        digitalWrite(m1, 0);
       }
```

```
    //Stop your motor
    else if (serialData == '3')
    {
     digitalWrite(m1, 0);
     digitalWrite(m2, 0);
    }

    //set Speed level 1
    else if (serialData == '4')
    {
      x=80;
    }

    //set Speed level 2
    else if (serialData == '5')
    {
      x=150;
    }
    //set Speed level 3
    else if (serialData == '6')
    {
      x=255;
    }
}
```

***How it works***

This arduino sketch assigns the digital pins 6,9,5 to be an output. This is where your dc motor driver pins are connected with. Arduino pins 6 and 9 are connected with IN1 and IN2 pins of motor driver. Arduino Pin 5 is connected to EN1 (Enable pin) of motor driver. This enable pin is set to analog value between 0 to 255 to control the speed of motor. In our case, we will only set to 170 and 255. (slow and full speed). As we seen in previous exercise, the incoming value from python is read in arduino via ***serial.read()*** and a conditional ***if_else*** statement is executed accordingly. For example: If the incoming value is '1', the DC motor is set to spin in clockwise direction.

***Verification***

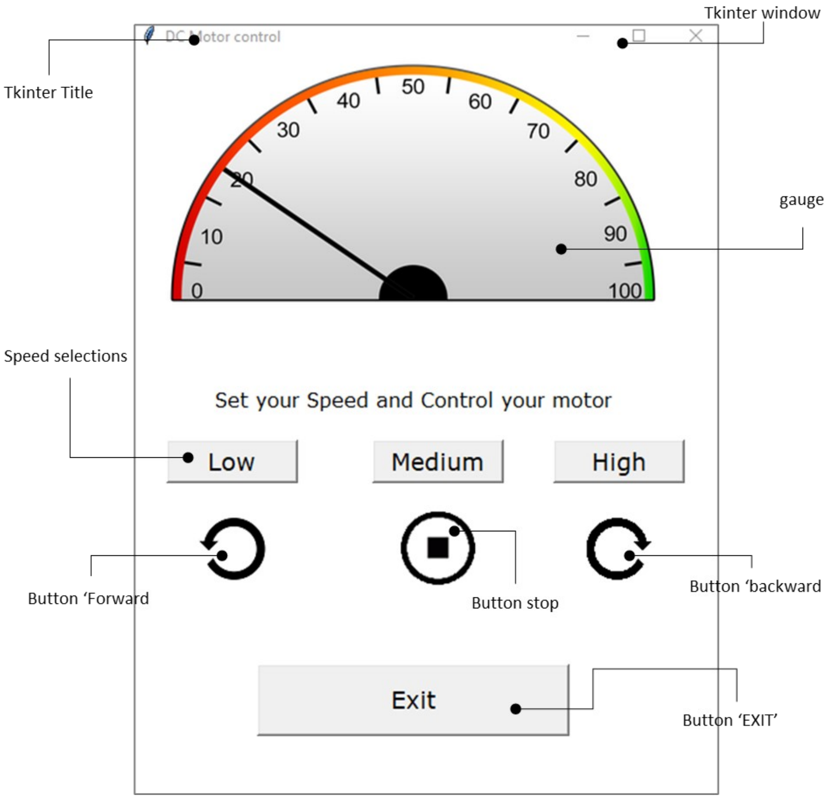
Once you compile and upload this sketch to your arduino, then try to send values from 1 to 5 via arduino IDE's ***serial monitor***. This should make your DC motor to perform respective actions based on inputs. If it works, then you are good to go with Python scripts for creating your GUI.

## Python script

Lets begin with Python script to build our GUI. As referred with previous exercises, we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tk-tools: a** package for creating ready made interactive GUI. Then establish a serial connection by assigning a respective COM ports. As described in previous exercise, create a tkinter window. Then we create respective buttons, dial gauge widget, scale widget and pack the them in tkinter window using grid method and place them in respective rows and columns.

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- *Connect your hardware according to circuit diagram provided.*
- *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Verify the program using Arduino IDE Serial monitor.*
- *Once verified, Close the Serial monitor of Arduino IDE.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! You will see your GUI opened.*



Set your Speed and Control your motor

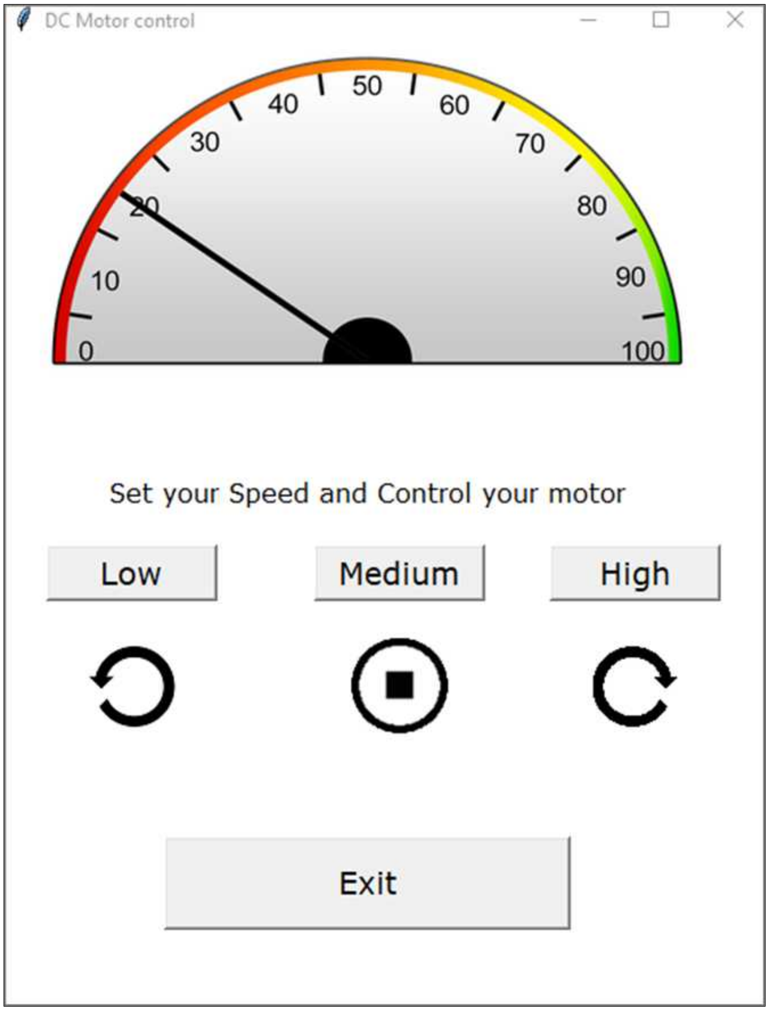
***Tkinter GUI***

## Python script

```
#Import following packages
import serial
import tkinter
import tk_tools
from PIL import Image, ImageTk

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial(‘com6',9600)

#Function call to close your program
def close_window():
    arduino.close()
    window.destroy()

#function call to set speed of motor
def set_speed(var):
    speed_gauge.set_value(var) #set the value for speed gauage
    print(var) #print value to cross-check
    value = int(var) #conver to int
    #condition statements
    if(value <=170 and value > 0):
        arduino.write(b'4') #write to arduino
    elif(value <=255 and value > 170):
        arduino.write(b'5') #write to arduino

#function call to forward button
def forward():
    arduino.write(b'1') #write to arduino

#function call to reverse button
def backward():
    arduino.write(b'2') #write to arduino

#function call to stop button
def stop():
    arduino.write(b'3') #write to arduino
```

```
# MAIN
window = tkinter.Tk() #create tkinter window
window.title("Servo angle control") #give title
window.configure(background="white") #change background color
window.geometry("500x700") #set size of tkinter window

#create frame are to place your buttons
f1 = tkinter.Frame(window)
f1.grid(row=2, column=0, sticky="nsew", pady=40)
f1.configure(background="white")

'''
Declare the images
Change the file path accordingly as per your folder
var =  ImageTk.PhotoImage(Image.open("<---your file path--->"))
'''
stopped = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/stop.png"))
left = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/left.png"))
right = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/right.png"))

# Create a silder slider to set speed of DC motor
motor = tkinter.Scale(window, activebackground="blue",
                label = "        Set the speed of your motor",
                bg = "white", font=('Verdana',14), from_=0, to=255,
                orient=tkinter.HORIZONTAL, length= 400,
                command = set_speed)
#create speed guage from tk-tools module
speed_gauge = tk_tools.Gauge(window, max_value=255, bg='white', label='Motor speed', unit=' Km/hr', height = 300, width = 500)

#Create label
label_text = tkinter.Label(f1, text="Control your Motor direction using the buttons below", font=('Verdana',12), bg="white", fg="#8B0000")

#create buttons
button_exit = tkinter.Button(window, text="Exit", font=('Verdana',14),
                     padx=100, pady = 10,
                     command=close_window)
```

```
#create you buttons
buttonback = tkinter.Button(f1, image = left, bg = "white", command= forward)
buttonquit = tkinter.Button(f1, image = stopped, bg = "white", command= stop)
buttonnext = tkinter.Button(f1, image = right, bg ="white", command= backward)

#pack your widgets
motor.grid(row=1, column=0)
speed_gauge.grid(row=0, column = 0)
label_text.grid(row=1, column=0,  columnspan =3)
button_exit.grid(row=6,column=0, columnspan =2)
buttonback.grid(row =3, column =0,pady =20, padx=50)
buttonquit.grid(row =3, column =1, pady =20, padx=50)
buttonnext.grid(row =3, column =2, padx=50, pady =20)

#execute the loop
window.mainloop()
```

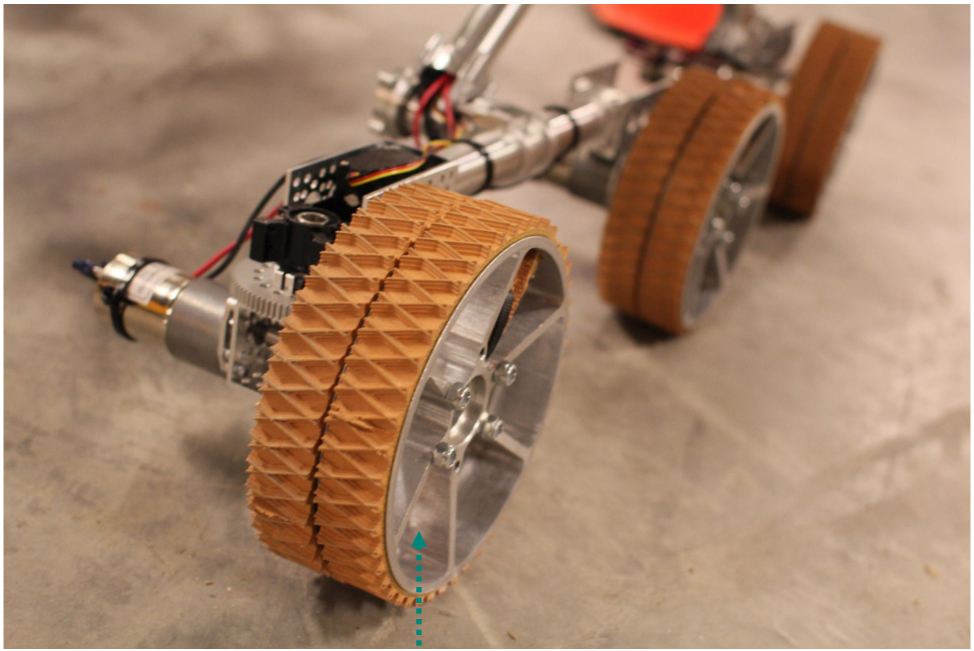


Set your Speed and Control your motor

***Python GUI Output***

## POTENTIAL APPLICATIONS

- Direction and speed control of mars Rover
- Direction and speed control of a Drones



Speed and direction control of Rover wheels

## CONCLUSION

In this chapter, we programmed buttons to switch between various speed levels and control the direction of a DC Motor. This is a first step needed to build your ROVERS.

# 8 Challenge-4

# PLOTTING SENSOR DATA

**Plot a sensor's data as a graph over a period of time using an Arduino micro controller with a help of script written in Python language. You will be creating a GUI consisting of a tkinter canvas widget, matplotlib library to plot graphs based on sensor's data. We will make use of Serial module from python to send or receive data from Arduino to our python GUI.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, matplotlib to plot a graph using a data of a sensor connected to a micro-controller. You will learn to create canvas, graph plot, arrange buttons on tkinter window, read a data from micro-controller and plot the data in real time graph.

Refer to GitHub repository mentioned in Page 139 for the codes:

***e4x_ping_sensor_data_Read.py***

***e4_ping_sensor_data_Read.ino***

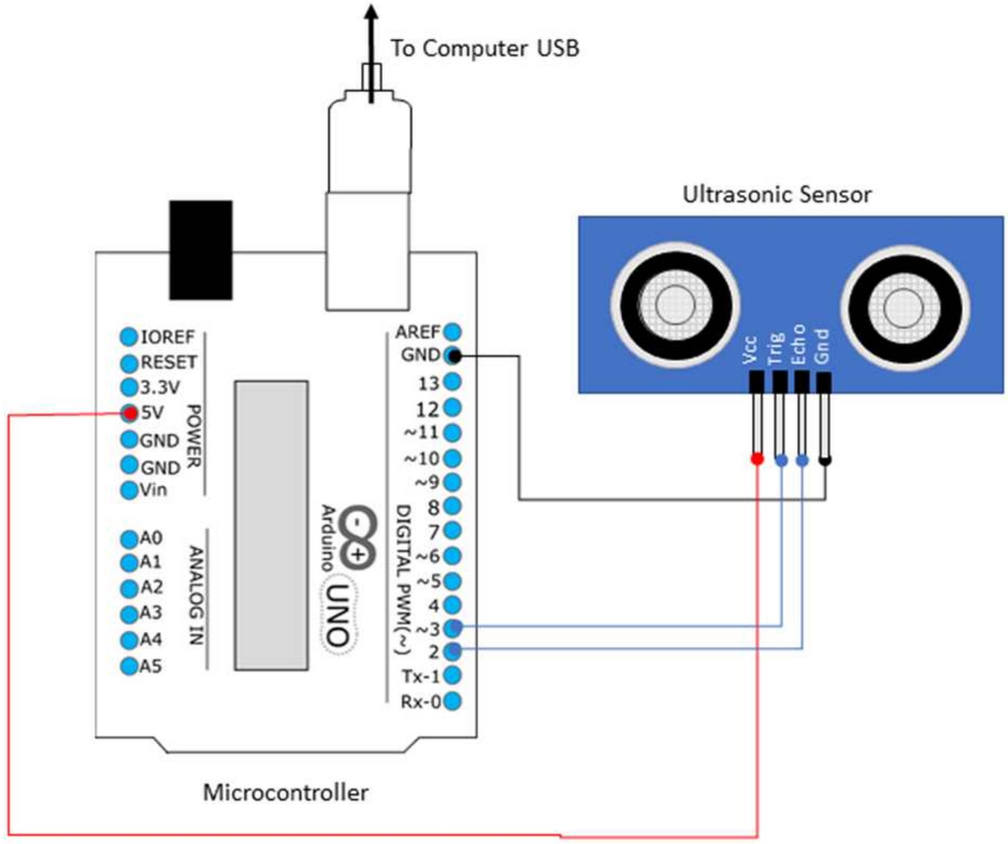
## Exercise

This exercise is about getting & plotting a real time data from a ping sensor connected to arduino micro-controller from a computer using a GUI built with Py-

## Hardware required

1. Arduino Uno – 1 Pc
2. Ping sensor (4 Pin Ultrasonic sensor) – 1 Pc
3. Jumper wires (F to M)- 4 Pcs
4. Bread Board—1 Pc (optional)

## Circuit schematic



**Circuit connections explainer:**

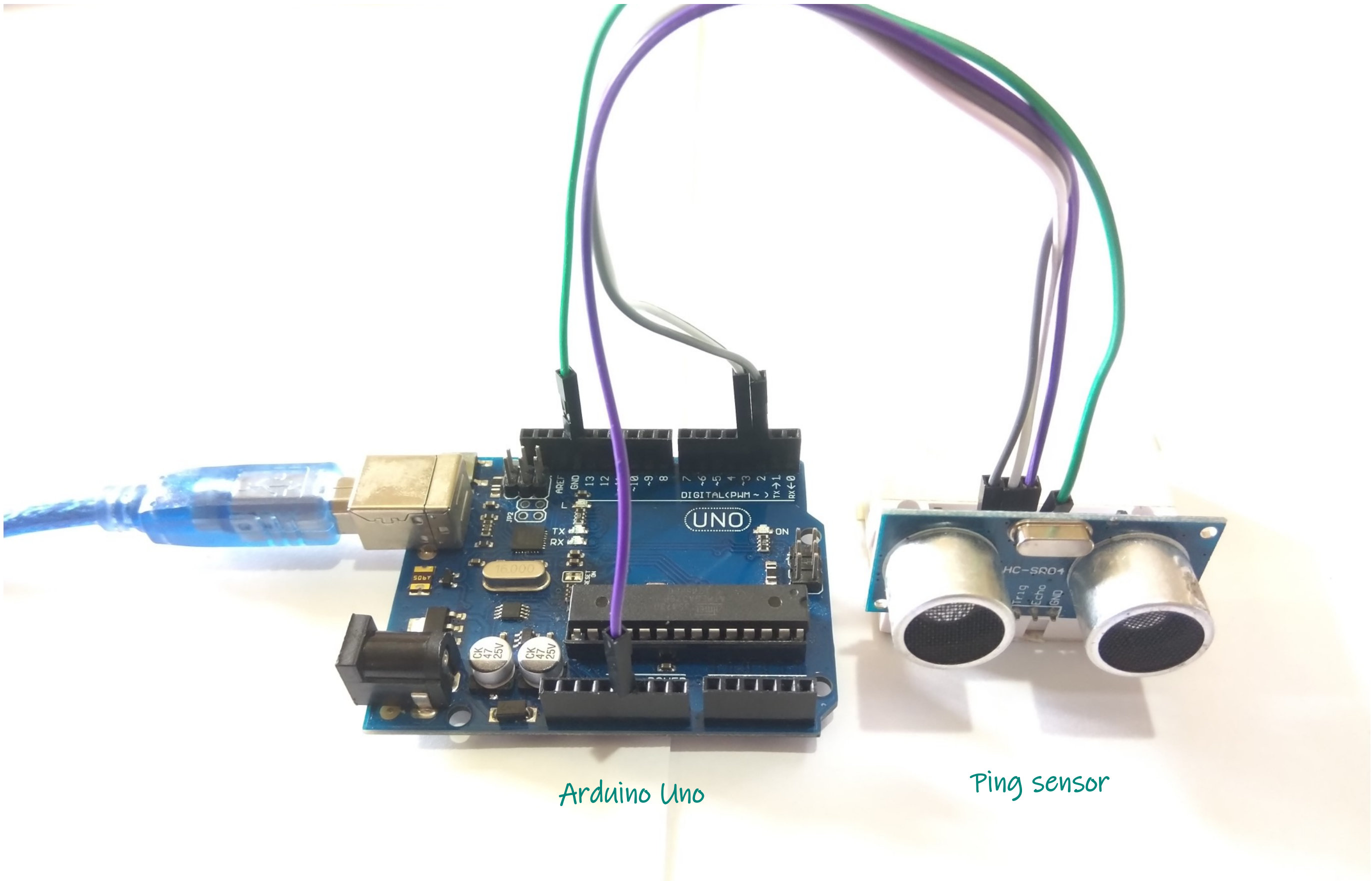
Ping sensor VCC PIN is connected to 5V PIN of arduino

Ping sensor TRIG PIN is connected to D3 PIN of arduino

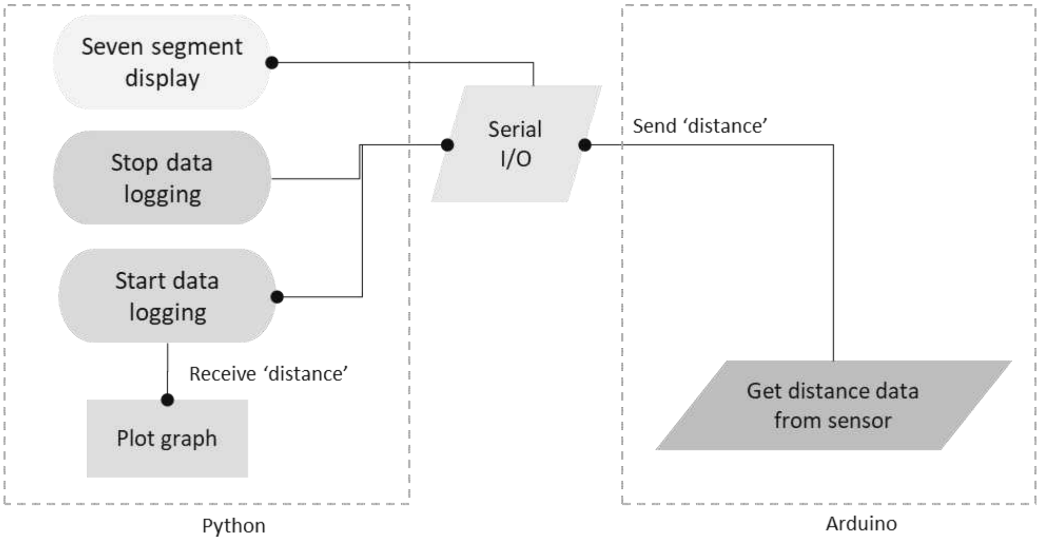
Ping sensor ECHO PIN is connected to D2 PIN of arduino

Ping sensor GND PIN is connected to GND PIN of arduino

# Hardware setup



## Work Flow



## Arduino script

```
//Declare variables
char  serialData;

//Ping sensor pins
int trigPin = 11;    // Trigger
int echoPin = 12;    // Echo
long duration, cm;

void setup() {
  //Serial Port begin
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.setTimeout(80);
}

void loop() {
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
```

```
 digitalWrite(trigPin, LOW);
 delayMicroseconds(5);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 // Read the signal from the sensor: a HIGH pulse whose
 // duration is the time (in microseconds) from the sending
 // of the ping to the reception of its echo off of an object.
 pinMode(echoPin, INPUT);
 duration = pulseIn(echoPin, HIGH);

 // Convert the time into a distance
 cm = (duration/2) / 29.1;     // Divide by 29.1 or multiply by 0.0343
 Serial.println(cm); //This will be sent to python for displaying graph
 delay(500);
}
```

***How it works***

This arduino sketch assigns the digital pins 10, 11 to be an output to trig and echo pin of your ultrasonic ping sensor. Then the trig pin is sent with HIGH pulse for 10 micro seconds followed by a LOW pulse for 5 micro seconds. Then the echo is read back and the time difference is calculated accordingly based on speed of sound and distance calculation. Then this data is printed over serial port for further use by our python script.

***Verification***

Once you compile and upload this sketch to your arduino, then try to get the distance data from arduino IDE's ***serial monitor***. Make sure to play around the sensor for various distance. If it works, then you are good to go with Python scripts for creating your GUI.

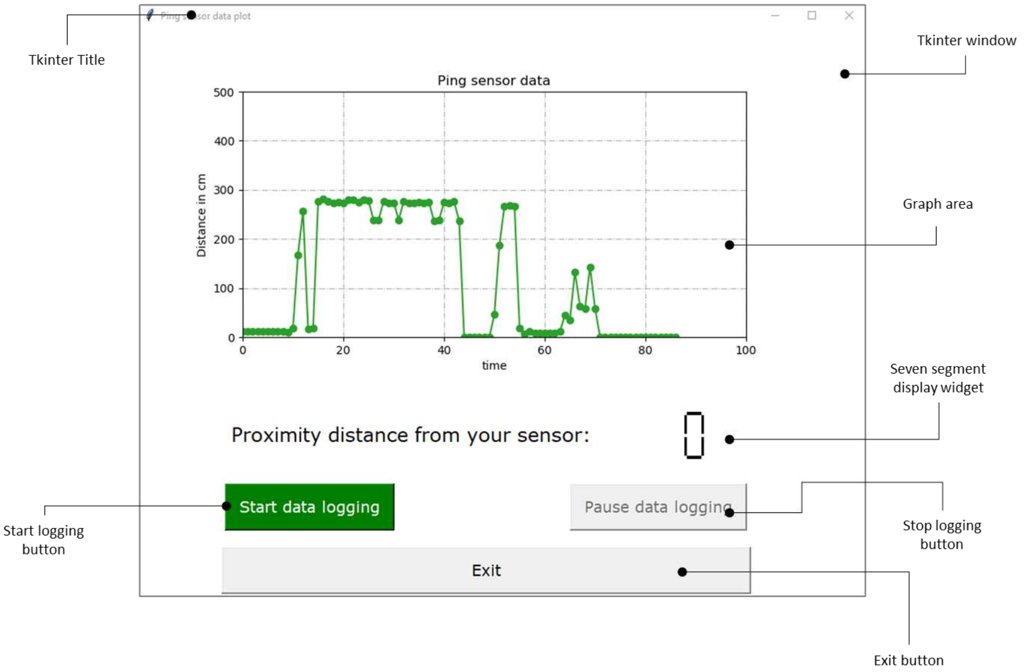
## Python script

Lets begin with Python script to build our GUI. As referred with previous exercises, we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tk-tools: a** package for creating ready made interactive GUI. Apart from this, we should also import ***matplotlib*** library as we are going to plot graphs in our tkinter canvas area. Other than this, we should also make use of ***nympy*** library as we are going to work on data arrays and list. Then establish a serial connection by assigning a respective COM ports. As described in previous exercise, create a tkinter window. Then we create respective buttons, a canvas widget to fix and plot the graph and a ***Seven segment display widget*** to display our sensor data. Arrange these widgets accordingly using a grid method.

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- ♦ *Connect your hardware according to circuit diagram provided.*
- ♦ *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- ♦ *Verify the program using Arduino IDE Serial monitor.*
- ♦ *Once verified, Close the Serial monitor of Arduino IDE.*
- ♦ *Now run your Python script from Spyder IDE.*
- ♦ *Ta da! You will see your GUI opened.*

***Tkinter GUI***



## Python script

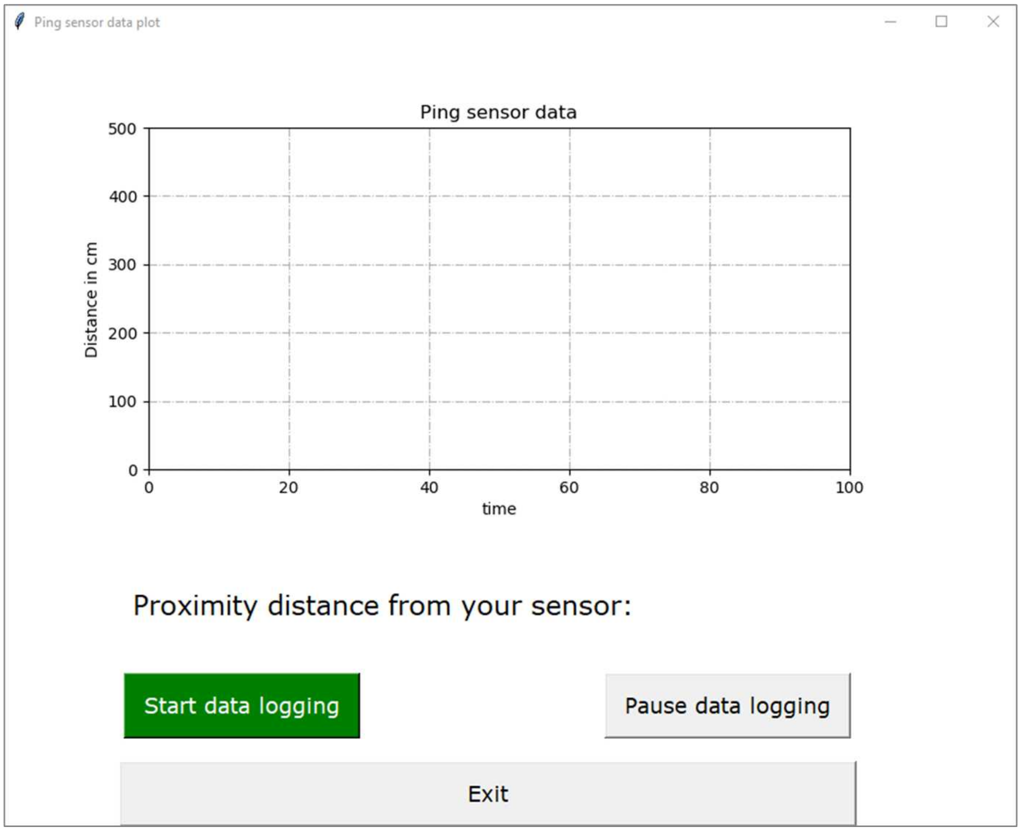
```
#Import following packages
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import tkinter
import numpy as np
import tk_tools
import serial

#Declare Global variables
data = np.array([])
condition = False

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial('COM6',9600);

#Plot your data
def plot_data():
    global condition, data #declare global variables again
    if (condition == True):
        a = arduino.readline() #read data from arduino
        b= int(float(a[0:len(a)-2].decode("utf-8"))) #convert to integer
        a.decode() #decode a for later use
        print(b)
        ss.set_value(str(b)) #set the value to seven segment display

        if(len(data) < 100):
            data = np.append(data,float(a[0:4]))
        else:
            data[0:99] = data[1:100]
            data[99] = float(a[0:4])
            lines.set_xdata(np.arange(0,len(data)))
            lines.set_ydata(data)
            canvas.draw()
```

```
# in after method 1 miliseconds
    # is passed i.e after seconds
    # main window will get plotted with data
    window.after(1,plot_data)

#start the plot
def plot_start():
    global condition
    condition = True
    start_button.config(state = 'disabled')
    stop_button.config(state = 'normal')
    arduino.reset_input_buffer()

#pause the data plot
def plot_stop():
    global condition
    condition = False
    stop_button.config(state = 'disabled')
    start_button.config(state = 'normal')

#exit function
def close_window():
    arduino.close()
    window.destroy()

#-Main GUI code
window = tkinter.Tk()#create tkinter window
window.title('Ping sensor data plot')#give title
window.configure(background = 'white')#set background color
window.geometry("900x700") # set the window size

#-create Plot object on GUI
fig = Figure(figsize=(8, 4), dpi=100);# add figure canvas
ax = fig.add_subplot(111)
```

```
#display only 100 samples
ax.set_title('Ping sensor data');
ax.set_xlabel('time')
ax.set_ylabel('Distance in cm')
ax.grid(True, linestyle='-.')
ax.set_xlim(0,100)
ax.set_ylim(0,500) #change this value according to max distance of your distance
lines = ax.plot([],[], color='C2',marker='o', markersize=6)[0]


canvas = FigureCanvasTkAgg(fig, master=window)  # A tkinter drawing area
canvas.get_tk_widget().grid(row=0,column=0, rowspan =2, columnspan =2, padx=30, pady=30)
canvas.draw()


#create buttons, other widgets
start_button = tkinter.Button(window, text = "Start data logging",
                              font=('Verdana',14),padx=10,
                              pady =10, bg = 'green', fg= 'white',
                              command = lambda: plot_start())



stop_button = tkinter.Button(window, text = "Pause data logging",
                             font=('Verdana',14), padx=10, pady =10,
                             command = lambda:plot_stop())


button_exit = tkinter.Button(window, text="Exit", font=('Verdana',14),
                             padx=300, pady =10,
                             command = close_window)


name_tag = tkinter.Label(window,
                         text= "Proximity distance from your sensor:                    ",
                         font=('Verdana',18), bg="White")


ss = tk_tools.SevenSegmentDigits(window, digits=4, background='white',
                                 digit_color='black', height=60)
```

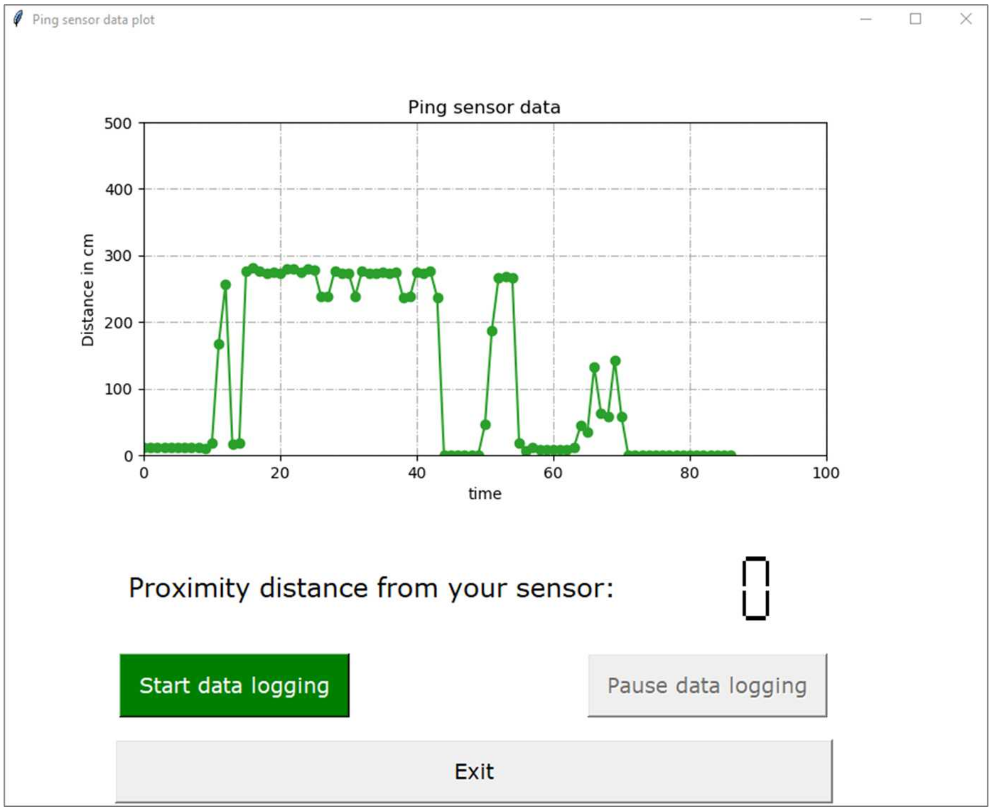
```
#pack all your widgets
start_button.grid(row=3,column=0, pady=20)
stop_button.grid(row=3,column=1, pady=20)
button_exit.grid(row=4,column=0, columnspan =2)
name_tag.grid(row=2, column=0, columnspan =2)
ss.grid(row=2, column=1, pady=10)

arduino.reset_input_buffer()
window.after(1,plot_data) #plot data every 0.001 second

#execute the loop
window.mainloop()
```

***Python GUI Output***



Inactive GUI

***Python GUI Output***



Proximity distance from your sensor: 0

Active GUI

## CONCLUSION

In this chapter we programmed to plot a sensor's data using Python and tkinter. We also displayed the sensor's output value in real time using tk_tools widget.

# 9

## Challenge-5

# SENSOR DATA LOGGING

**Plot a sensor's data as a graph over a period of time using an Arduino micro controller with a help of script written in Python language and also save a log of data by exporting to csv format. You will be creating a GUI consisting of a tkinter canvas widget, matplotlib library to plot graphs based on sensor's data and exporting to csv file. We will make use of Serial module from python to send or receive data from Arduino to our python GUI and csv module to work on CSV files.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, CSV, datetime, matplotlib to plot a graph using a data of a sensor connected to a micro-controller and then save the data into csv file. You will learn to create canvas, graph plot, arrange buttons on tkinter window, read a data from micro-controller and plot the data in real time graph.  Then simultaneously write a code to create, open and save a data to csv file.

Refer to GitHub repository mentioned in Page 139 for the codes:

***ex5_temperature_logging.py***

***e5_temperature_logging.ino***

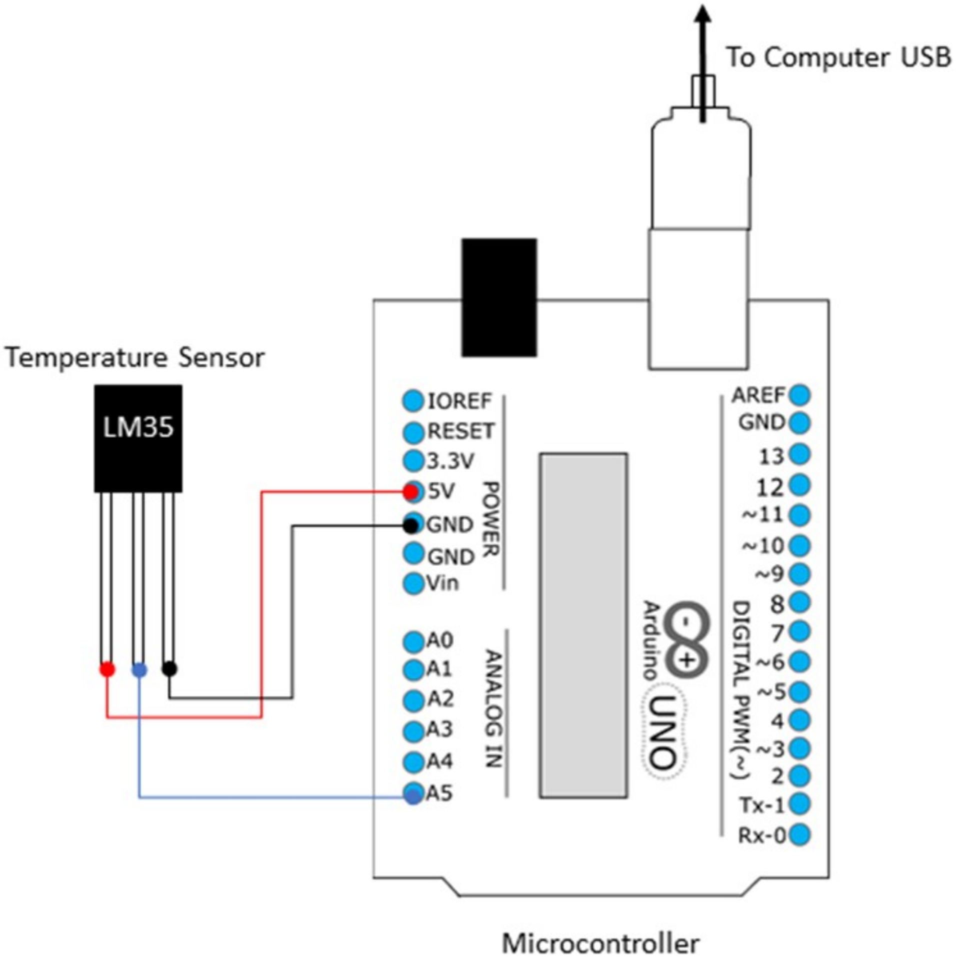
## Exercise

This exercise is about getting & plotting a real time data from a Temperature sensor connected to arduino micro-controller from a computer using a GUI built

## Hardware required

1. Arduino Uno – 1 Pc
2. Temperature sensor (2 pin sensor) – 1 Pc
3. Jumper wires (F to M)- Few Pcs
4. 10K ohm resistor—1 Pc
5. Bread Board—1 Pc (optional)

**Cir-**



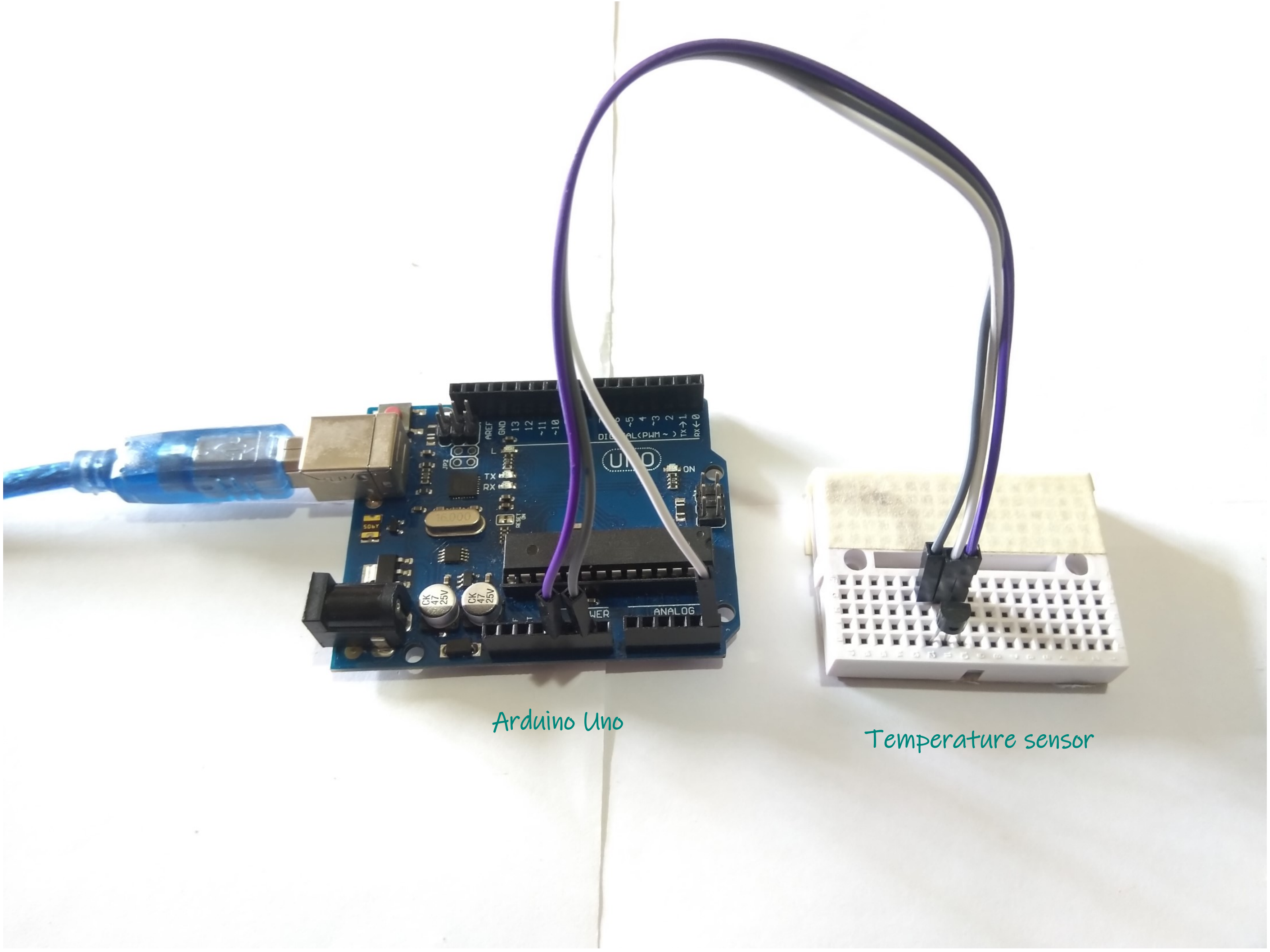
**Circuit connections explainer:**

LM35 sensor VCC PIN is connected to 5V PIN of arduino

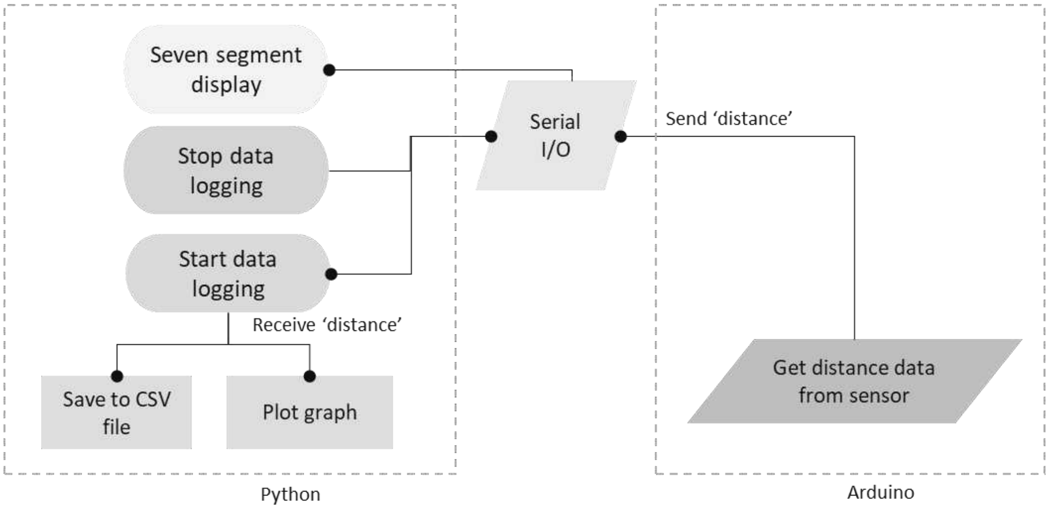
LM35 sensor ANALOG OUT PIN is connected to A5 PIN of arduino

LM35 sensor GND PIN is connected to GND PIN of arduino

# Hardware setup



## Work Flow



## Arduino script

```
//Declare variables
char  serialData;
const int sensorPin= 0; //Sensor pin connects to analog pin A0
int data;           //the variable that will hold the temperature value reading

void setup()
{
Serial.begin(9600); //sets the baud rate at 9600 so we can check the values the sensor is obtaining on the Serial Monitor
Serial.setTimeout(80);
}

void loop()
{
  data= analogRead(sensorPin); //the sensor takes readings from analog pin A0
  float value= ( data/1024.0)*5000;
  float Celsius = value/10;

   #Use the following line if necessary
  #int Celsius = data - 546;  //adjust this value accordingly to your sensor

  Serial.println(Celsius ); //send this data to python
}
```

***How it works***

This arduino sketch assigns the analog PIN 0 of arduino to get the input from Temperature sensor. This data is read by micro controller and adjusts the analog value to readable temperature unit. Then this data is printed over serial port for further use by our python script.

***Verification***

Once you compile and upload this sketch to your arduino, then try to get the temperature data from arduino IDE's ***serial monitor***. Make sure to play around the sensor for changes in temperature. If it works, then you are good to go with Python scripts for creating your GUI. You may need to spend a bit of time in adjusting your temperature measurements.

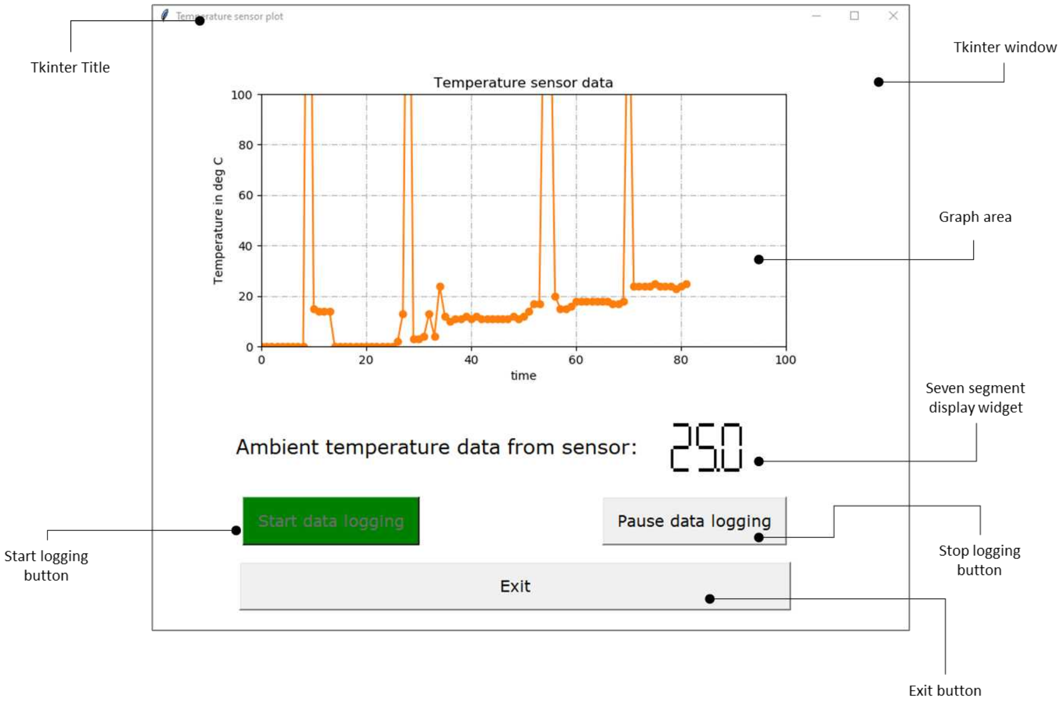
## Python script

Lets begin with Python script to build our GUI. As referred with previous exercises, we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tk-tools: a** package for creating ready made interactive GUI. Apart from this, we should also import ***matplotlib*** library as we are going to plot graphs in our tkinter canvas area. Other than this, we should also make use of ***nympy*** library as we are going to work on data arrays and list. Then establish a serial connection by assigning a respective COM ports. As described in previous exercise, create a tkinter window. Then we create respective buttons, a canvas widget to fix and plot the graph and a ***Seven segment display widget*** to display our sensor data. Arrange these widgets accordingly using a grid method.

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- *Connect your hardware according to circuit diagram provided.*
- *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Verify the program using Arduino IDE Serial monitor.*
- *Once verified, Close the Serial monitor of Arduino IDE.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! You will see your GUI opened.*

***Tkinter GUI***



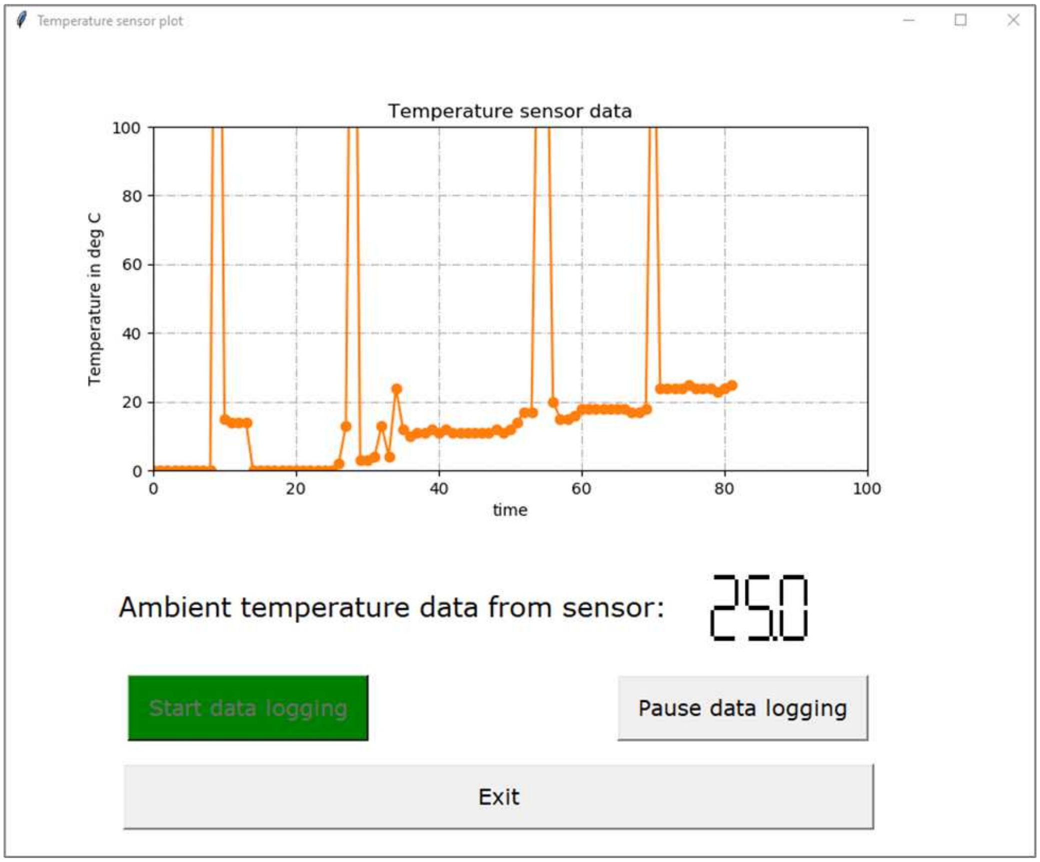
## Python script

```
#Import following packages
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import tkinter
import numpy as np
import tk_tools
import serial
import csv
import datetime

#Declare Global variable
data = np.array([])
condition = False

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial('COM6',9600);
```

```
#Plot your data
def plot_data():
    global condition, data #declare global variables again

    if (condition == True):

        a = arduino.readline() #read data from arduino
        b= float(a[0:len(a)-2].decode("utf-8"))  #convert to integer
        a.decode() #decode a for later use
        print(b)
        ss.set_value(str(b))  #set the value to seven segment display

        #Open a csv file and save data to it
        with open("test_data.csv","a", newline='') as f:
            writer = csv.writer(f,delimiter=",")
            writer.writerow([datetime.datetime.strftime(datetime.datetime.now(), '%
Y, %m, %d, %H, %M, %S'),str(b)])

        if(len(data) < 100):
            data = np.append(data,float(a[0:4]))
        else:
            data[0:99] = data[1:100]
            data[99] = float(a[0:4])

        lines.set_xdata(np.arange(0,len(data)))
        lines.set_ydata(data)
        canvas.draw()

    # in after method 1 miliseconds
    # is passed i.e after seconds
    # main window will get plotted with data
    window.after(1,plot_data)

#start the plot
def plot_start():
    global condition
    condition = True
    start_button.config(state = 'disabled')
    stop_button.config(state = 'normal')
    arduino.reset_input_buffer()
```

```
#pause the data plot
def plot_stop():
    global condition
    condition = False
    stop_button.config(state = 'disabled')
    start_button.config(state = 'normal')

#exit function
def close_window():
    arduino.close()
    window.destroy()



#MAIN LOOP
window = tkinter.Tk() #create tkinter window
window.title('Temperature sensor plot') #give title
window.configure(background = 'white') #set background color
window.geometry("900x720") # set the window size

#-create Plot object on GUI
# add figure canvas
fig = Figure(figsize=(8, 4), dpi=100);
ax = fig.add_subplot(111)

#display only 100 samples
ax.set_title('Temperature sensor data');
ax.set_xlabel('time')
ax.set_ylabel('Temperature in deg C')
ax.grid(True, linestyle='-.')
ax.set_xlim(0,100)
ax.set_ylim(0,100)
lines = ax.plot([],[],color='C1',marker='o', markersize=6)[0]

canvas = FigureCanvasTkAgg(fig, master=window)  # A tk.DrawingArea.
canvas.get_tk_widget().grid(row=0,column=0, rowspan =2, columnspan =2,
padx=30, pady=30)
canvas.draw()
```

```
#create buttons, other widgets
start_button = tkinter.Button(window, text = "Start data logging",
                              font=('Verdana',14),padx=10, pady =10,
                              bg = 'green', fg= 'white',
                              command = lambda: plot_start())



stop_button = tkinter.Button(window, text = "Pause data logging",
                             font=('Verdana',14), padx=10, pady =10,
                             command = lambda:plot_stop())


button_exit = tkinter.Button(window, text="Exit", font=('Verdana',14),
                             padx=300, pady =10,
                             command = close_window)



name_tag = tkinter.Label(window, text= "Ambient temperature data from sensor:                        ",
                         font=('Verdana',18), bg="White")


ss = tk_tools.SevenSegmentDigits(window, digits=4, background='white',
                                 digit_color='black', height=60)


#pack your widgets
start_button.grid(row=3,column=0, pady=20)
stop_button.grid(row=3,column=1, pady=20)
button_exit.grid(row=4,column=0, columnspan =2)
name_tag.grid(row=2, column=0, columnspan =2)
ss.grid(row=2, column=1, pady=10)


arduino.reset_input_buffer()


window.after(1,plot_data)


#execute the loop
window.mainloop()
```



***Python GUI Output***

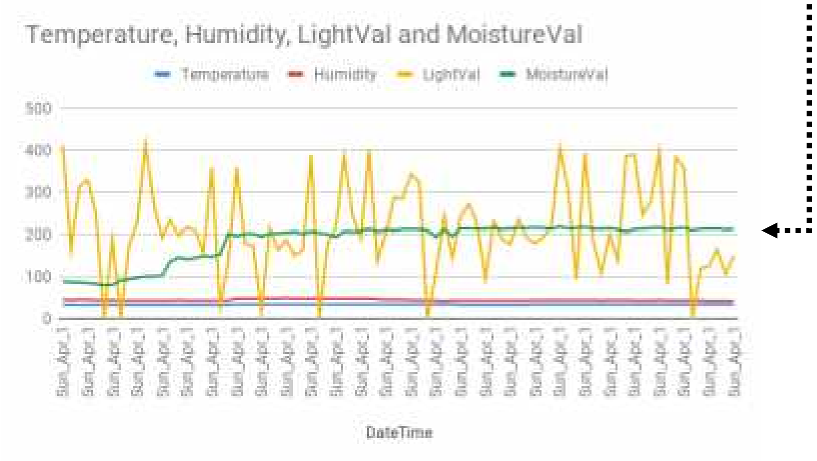
***Temp log saves into test_data.csv file***

test_data.csv

## POTENTIAL APPLICATIONS

- IoT plant sensor real time data logging and Saving to a file
- Self driving Robot sensor data logging and analyses

IOT Plant sensor



Sensor data logged

## CONCLUSION

In this chapter, we learned to log our sensor's data over a time period and save it to a file in our computer. This is an additional step as compared to our previous challenge. This can be extended  to new possibilities in Wireless sensing and IOT.

# 10 Challenge-6

# CONTROL OF RGB LED

**Pick and set a color to a RGB LED connected to an Arduino micro controller with a help of script written in Python language. You will be creating a GUI consisting of a tkinter color picker widget. We will make use of Serial module from python to send or receive data from Arduino to our python GUI and set a color to a RGB led.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, color picker widget to assign a color value to a RGB LED connected to a micro-controller.

Refer to GitHub repository mentioned in Page 139 for the codes:

***ex6_rgb_led_control.py***

***e6_rgb_led_control.ino***

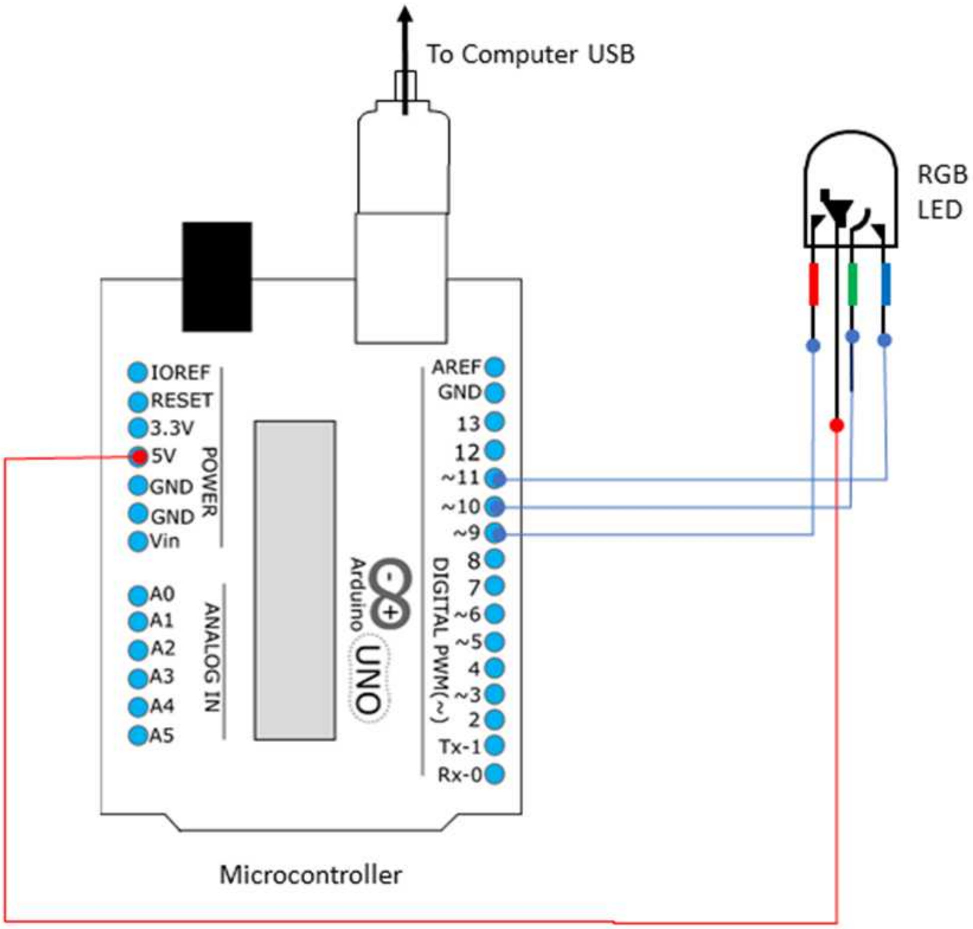
## Exercise

This exercise is about a control of RGB LED connected to arduino micro-controller from a computer using a GUI built with Python script.

## Hardware required

1. Arduino Uno – 1 Pc
2. RGB LED – 1 Pc
3. Jumper wires (F to M)- Few Pcs
4. Bread Board—1 Pc (optional)

## Circuit schematic



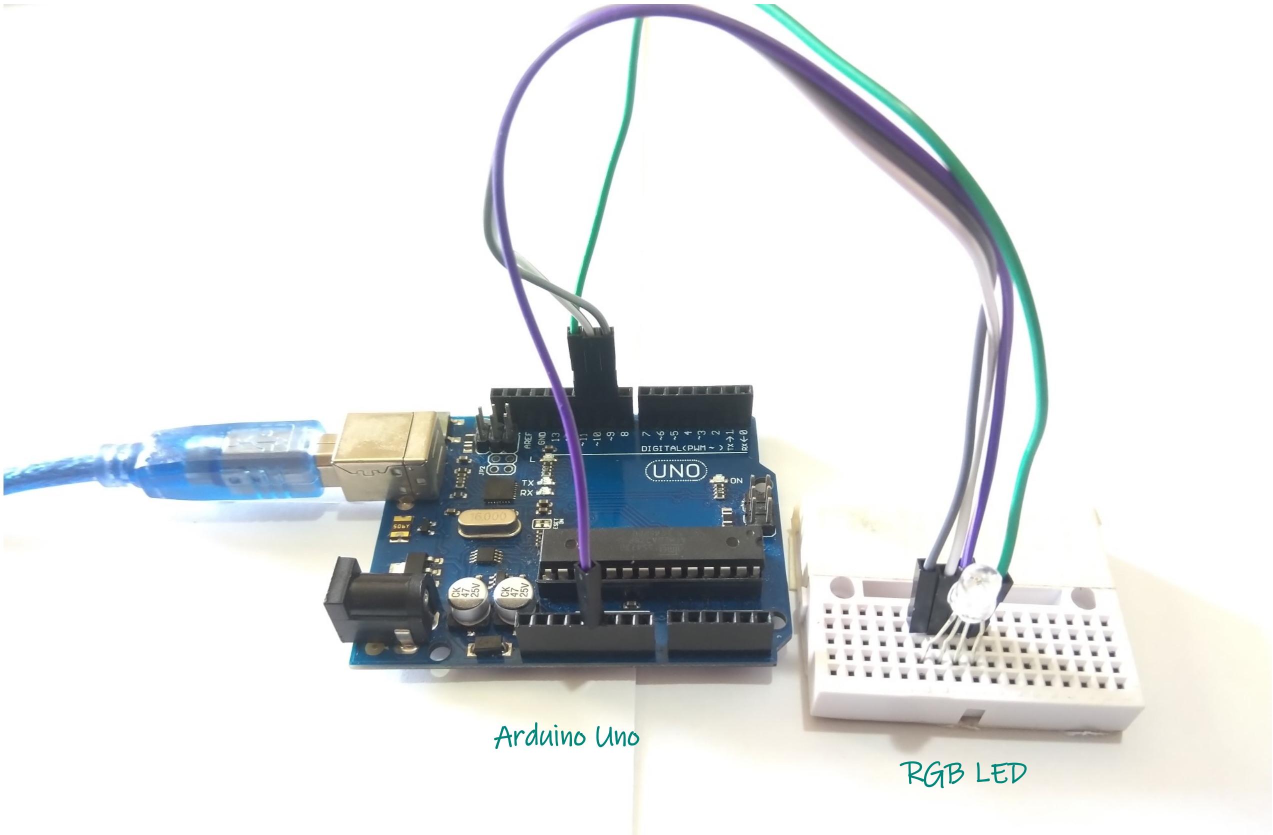
**Circuit connections explainer:**

RGB LED ANODE (Long leg) PIN is connected to 5V PIN of arduino

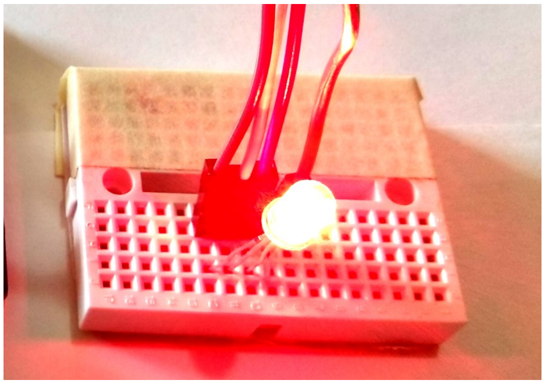
RGB LED RED PIN (1st Pin) is connected to D11 PIN (PWM pin) of arduino

RGB LED GREEN PIN (3rd Pin) is connected to D10 PIN (PWM pin) of arduino

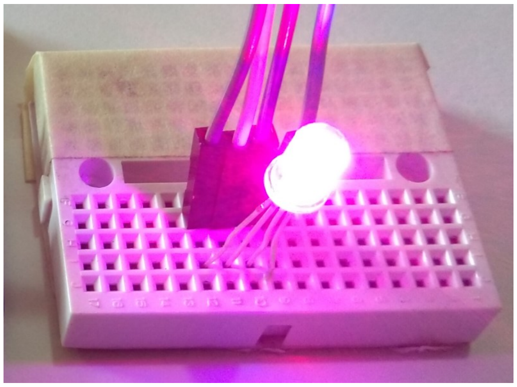
RGB LED BLUE PIN (4th Pin) is connected to D9 PIN (PWM pin) of arduino



## Hardware setup

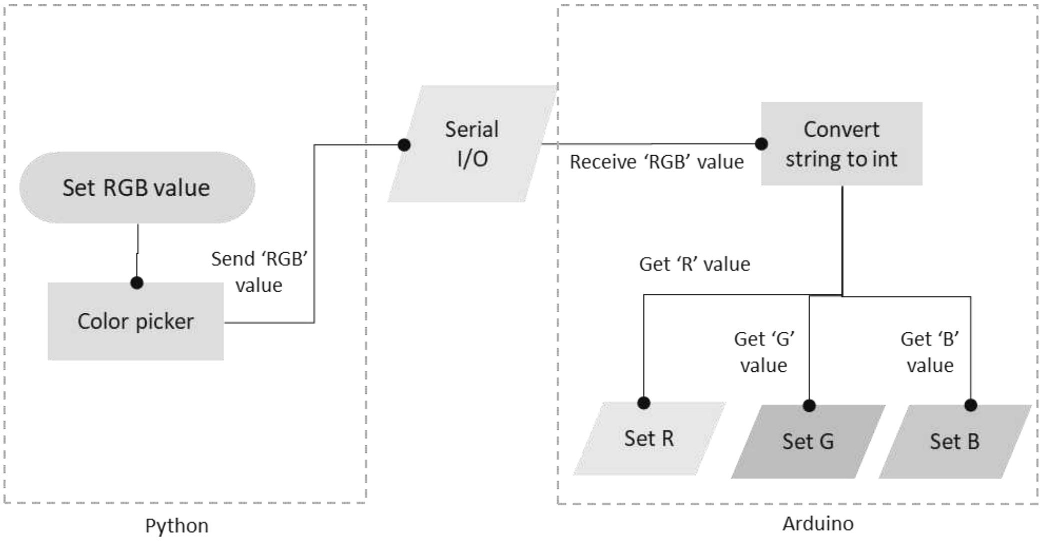


RED Color of RGB LED



PURPLE Color of RGB LED

## Work Flow



## Arduino script

```
//Declare variables
int incomingByte = 0; // for incoming serial data
String myString;
long myStringInt = 0;

//Set RGB led pins. Mke sure its connected to PWM pins of arduino
int redPin   = 9; //Red led connected to arduino PWM pin
int greenPin = 10; //green led connected to arduino PWM pin
int bluePin  = 11; //blue led connected to arduino PWM pin

//Setup your baudrate & arduino pins
void setup()
{
  pinMode(redPin,   OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin,  OUTPUT);
  Serial.begin(9600);
  Serial.setTimeout(10);  // improves the serial response time
  //set initial color of your LED to red
  analogWrite(redPin, 0);
  analogWrite(greenPin, 255);
  analogWrite(bluePin, 255);
}
```

```
void loop()
{
  myString = getSerial(); //get serial functions gets the data from python
  if (myString.startsWith("#"))
  {
    String value;
    myString.remove(0, 1);      // removes "#" to only have the numbers
    Serial.println(myString);   // print this value for cross-checking
    char charbuf[8];
    myString.toCharArray(charbuf,8);
    long int rgb= strtol(charbuf,0,16); //=>rgb=0x001234FE;

    //Convert values for red, green, blue from hex value
    byte r=(byte)(rgb>>16);
    byte g=(byte)(rgb>>8);
    byte b=(byte)(rgb);

    //Set your RGB led to respective analog value
    Serial.println(r); // print this value for cross-checking
    Serial.println(g); // print this value for cross-checking
    Serial.println(b); // print this value for cross-checking

    //As LED's are connected to PWM pins, you need to subtract from 255
    // (255, 0, 0) --> RED color is mapped --> (0,255,255) for PWM pins
    analogWrite(redPin, 255-r); //Subtract value from 255 as its PWM pin
    analogWrite(greenPin, 255-g);
    analogWrite(bluePin, 255-b);
  }
}
//function to get data from python
String getSerial()
{
  String a;
  while (Serial.available())
  {
    a = Serial.readString();
    Serial.println(a);
    return (a);
  }
}
```

***How it works***

This arduino sketch assigns the Digital PIN 9, 10, 11 of arduino to control the RGB LED values. Make sure that it has to be connected to PWM pins in order to control the intensity of RGB led. A serial data consisting of RGB values in string format is read by micro controller from python and converts to long int type and assigns respective analog values to individual R, G, B LED's.

***Verification***

Once you compile and upload this sketch to your arduino, then try to send a RGB value in hexadecimal format (Ex: #FFFF00) from arduino IDE's ***serial monitor***. If it works, then you are good to go with Python scripts for creating your GUI. You may need to spend a bit of time in adjusting your temperature measurements.

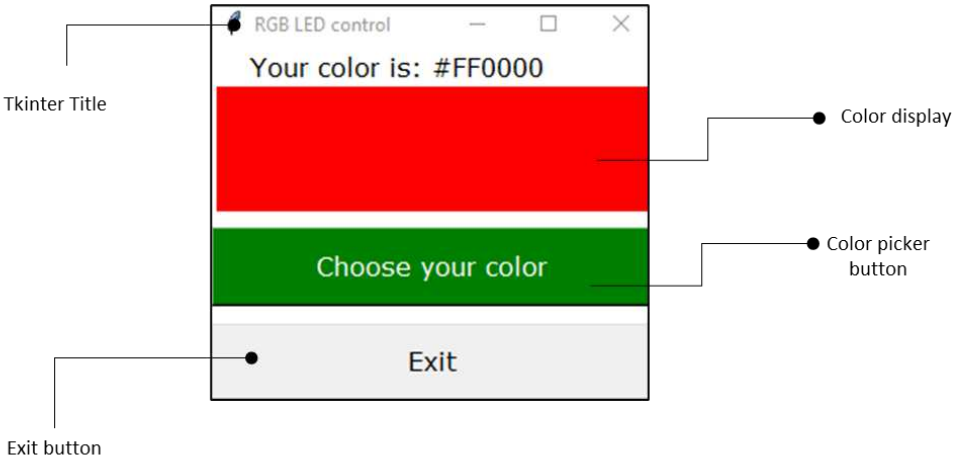
## Python script

Lets begin with Python script to build our GUI. As referred with previous exercises, we should import respective modules to be used in our script. This includes **Pyserial**: for establishing serial port communication with external devices (in our case, its arduino Uno connected to computer via USB), **tkinter:** an interface for creating GUI widgets, **tkcolorpicker**: **a** package for using a color palette and choosing a color from it. This color picker opens a pop up color palette and asks you to choose a color. Then the chosen color can be subscribed and used for other purpose. Then establish a serial connection by assigning a respective COM ports using **PySerial** module. As described in previous exercise, create a tkinter window. Then we create respective buttons, a label/button to show the color the user picked and a link to open color picker. Arrange these widgets accordingly using a grid method.

***Execution:*** Once everything is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- ♦ *Connect your hardware according to circuit diagram provided.*
- ♦ *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- ♦ *Verify the program using Arduino IDE Serial monitor.*
- ♦ *Once verified, Close the Serial monitor of Arduino IDE.*
- ♦ *Now run your Python script from Spyder IDE.*
- ♦ *Ta da! You will see your GUI opened.*

*Tkinter GUI*



Tkinter Title

Color display

Color picker button

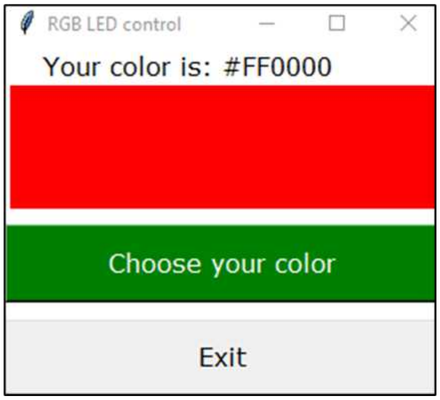
Exit button

## Python script

```
import serial
import tkinter
from tkcolorpicker import askcolor

#Connect to arduino via serial port
#com6 --> Change port accordingly to yours
arduino = serial.Serial('com6',9600)

def close_window(): #exit the window
    arduino.close()
    window.destroy()

def led_on():
    cc = askcolor((255, 0, 0), window) #open color picker and get color
    r = cc[0][0] #get red value from tuple
    g = cc[0][1] #get green value from tuple
    b = cc[0][2] #get blue value from tuple
    c= cc[1] #get the hex value from tuple
    color_label.config(bg= str(c))
    label.config(text= "Your color is:" + str(c))
    print(r,g,b) #print value for cross-check
    print(c)   #print value for cross-check
    arduino.write(str(c).encode()) #write hex value to arduino
```

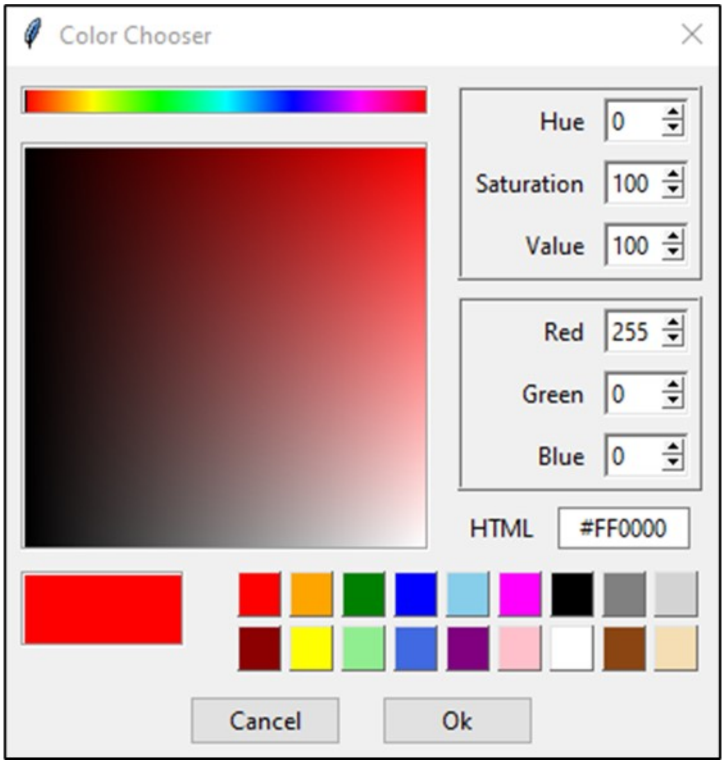
```
# MAIN
window = tkinter.Tk() #create tkinter window
window.title("RGB LED control") #create title
window.configure(background="white") #set background color

#create widgets
button_on = tkinter.Button(window, text="Choose your color",
                           font=('Verdana',12), padx=60, pady =10,
                           bg="green",fg="white",
                           command = led_on)

label = tkinter.Label(window, text="Your color is: #FF0000",
                      font=('Verdana',12),fg="black", bg ='white')

color_label = tkinter.Label(window,padx=135, pady =30, bg="red", fg="black")

button_exit = tkinter.Button(window, text="Exit", font=('Verdana',12),
                             padx=120, pady =10,
                             command = close_window)

#pack widgets
button_on.grid(row=2,column=0, columnspan =2, pady=10)
label.grid(row=0,column=0)
color_label.grid(row=1,column=0, columnspan =2)
button_exit.grid(row=4,column=0, columnspan =2)

#execute the loop
window.mainloop()
```

***Python GUI Output***



Your color is: #FF0000

Before color picker selection



Hue 0

Saturation 100

Value 100

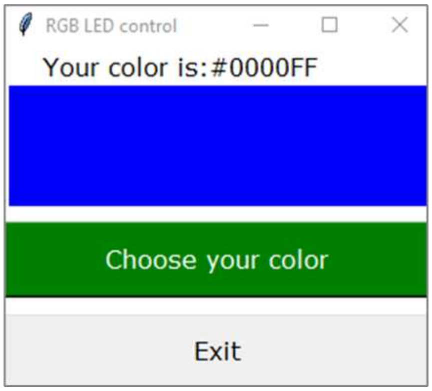
Red 255

Green 0

Blue 0

HTML #FF0000

Color picker



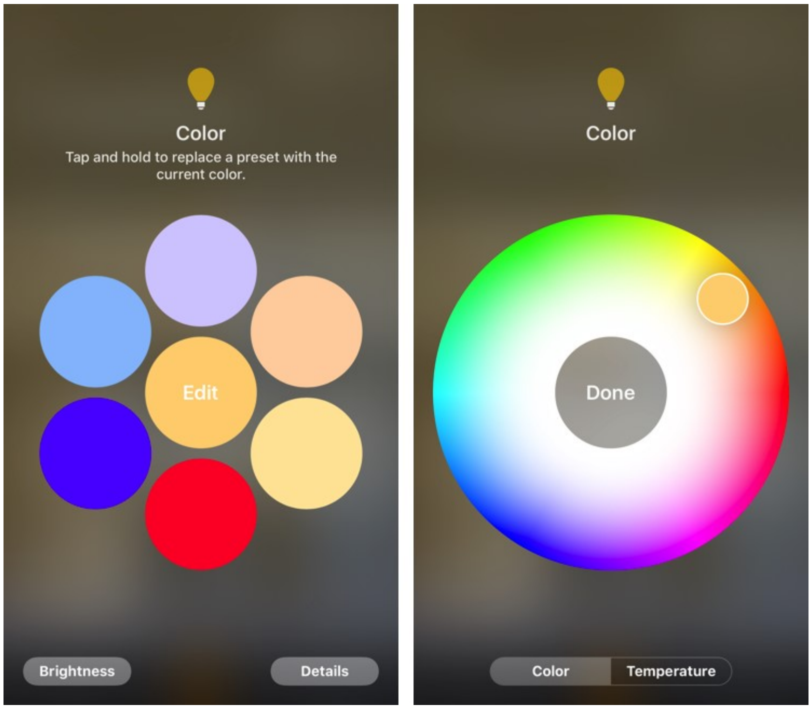
Your color is:#0000FF

After color picker selection

## POTENTIAL APPLICATIONS

- Wireless control of RGB lamp in your living room by picking your favourite Color

RGB LED light



Color

Tap and hold to replace a preset with the current color.

GUI app to control color of lights

## CONCLUSION

In this chapter, we programmed an interactive color picker for setting a color to a RGB LED. Now, you can build a RGB lamp for your living room.

# 11

## Challenge-7

# WIRELESS CONTROL OF ARDUINO

**Send data to arduino micro controller from Python with a help of a Bluetooth module connected with Arduinio. We will make use of Serial module from python to send or receive data from Arduino to our python GUI and control an LED but in a  wireless way.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, tkinter-tools to control the state of an LED connected to a micro-controller in wireless way using Bluetooth.

Refer to GitHub repository mentioned in Page 139 for the codes:

***ex7_bluetooth_python_itnterface.py***

***e7_bluetooth_python_itnterface.ino***

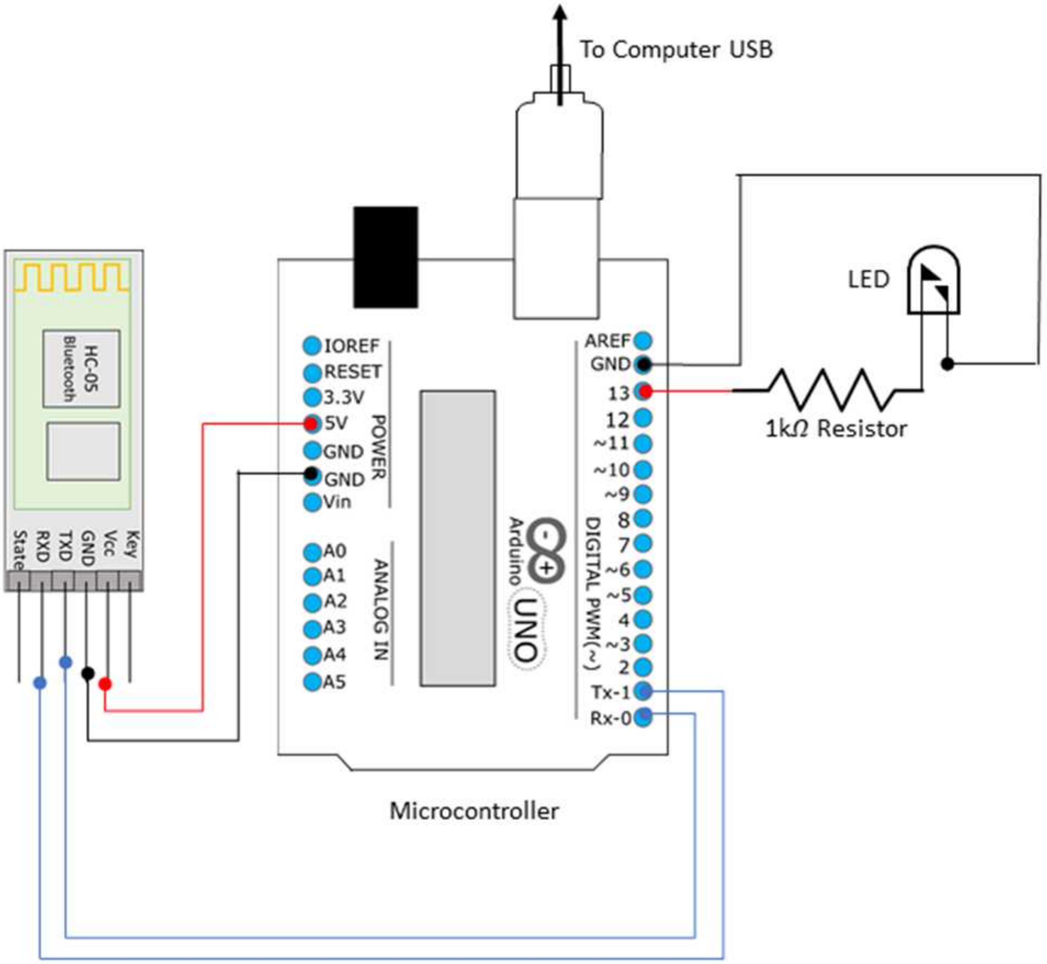
## Exercise

This exercise is about a wireless control of LED connected to arduino micro-controller from a computer using a GUI built with Python script with Bluetooth.

## Hardware required

1. Arduino Uno – 1 Pc
2. LED – 1 Pc
3. Jumper wires (F to M)- Few Pcs
4. Bread Board—1 Pc (optional)
5. Bluetooth (HC-05)—1 Pc

## Circuit schematic



**Circuit connections explainer:**

LED's anode (longer leg) is connected to PIN 13 of arduino

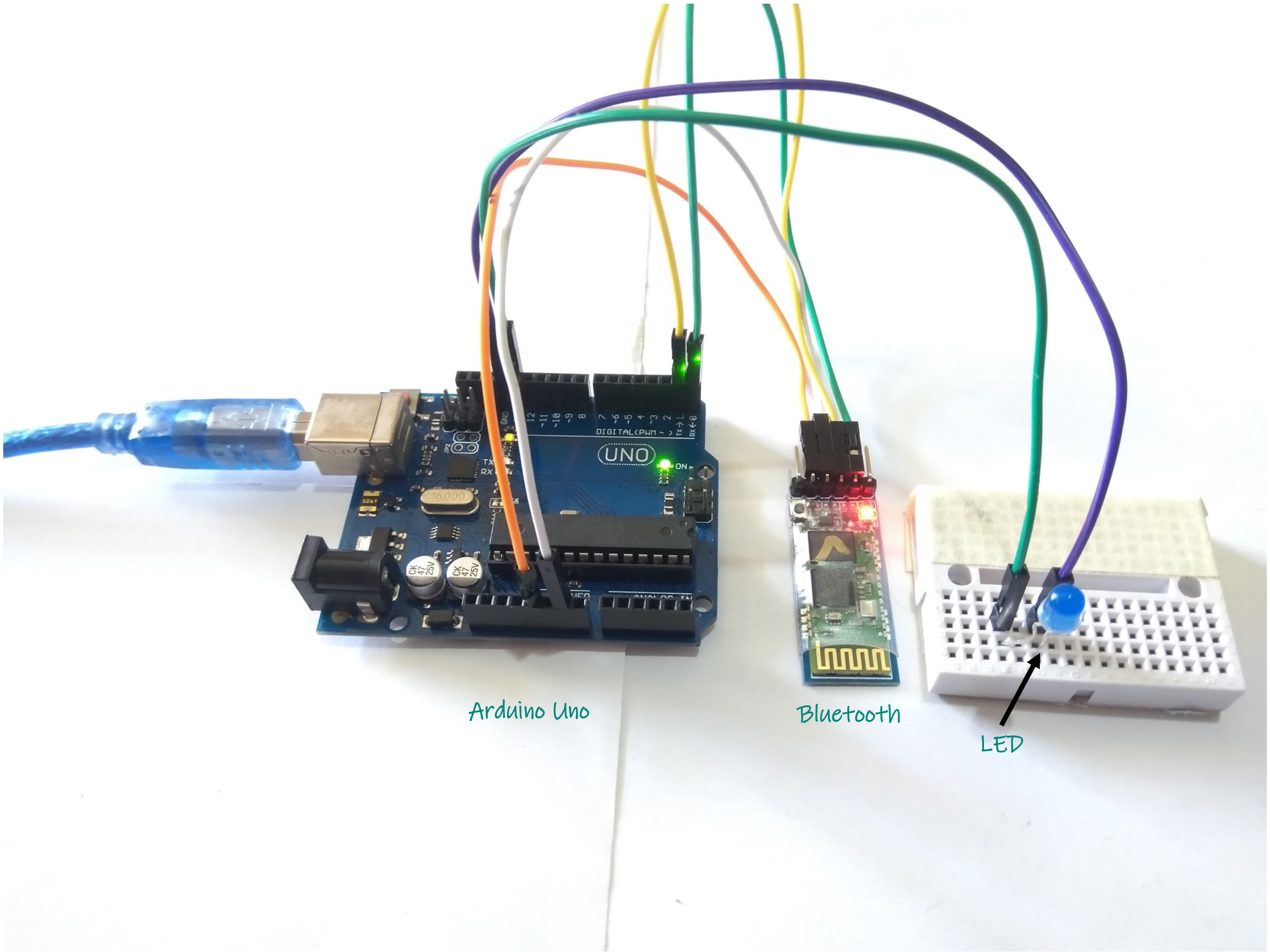
LED's cathode (shorter leg) is connected to GND of arduino via a resistor

Bluetooth TX PIN is connected to RX PIN of arduino

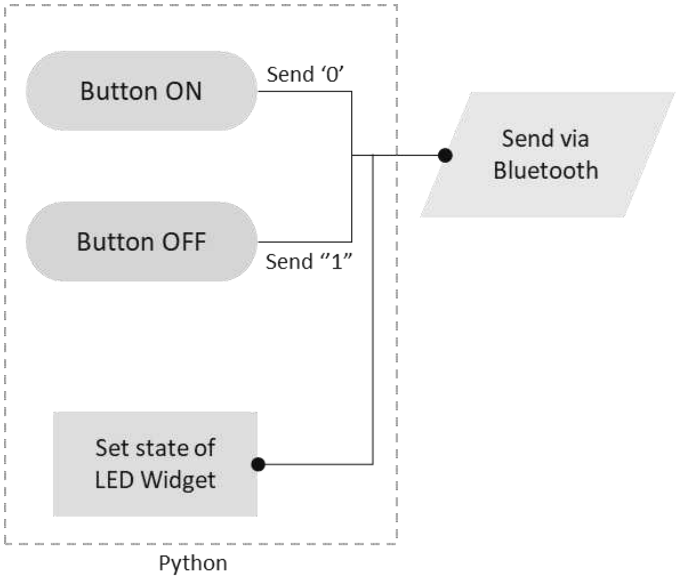
Bluetooth RX PIN is connected to TX PIN of arduino

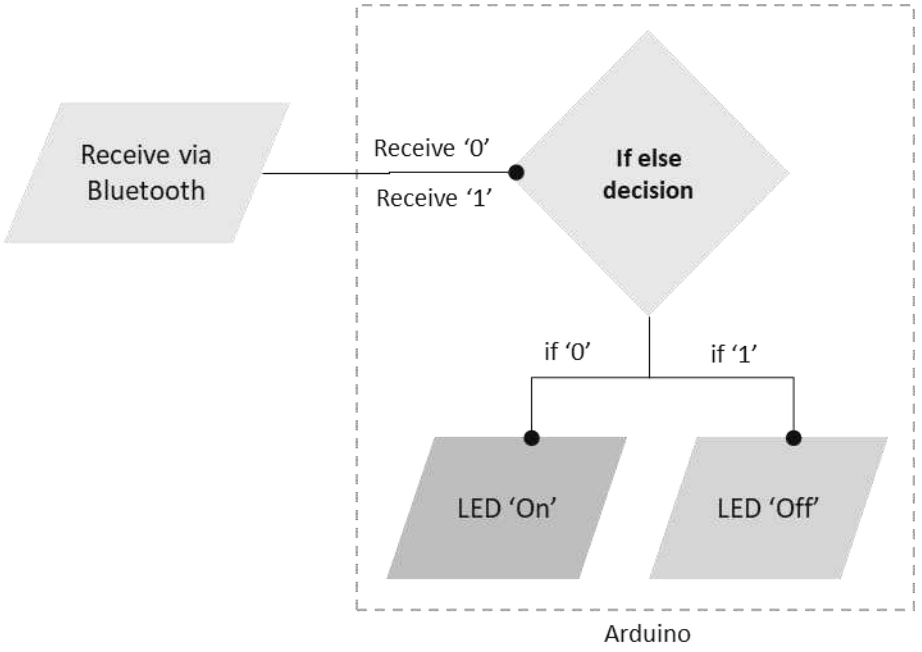
Bluetooth VCC / GND PIN is connected to 5V / GND PIN of arduino accordingly

# Hardware setup



## Work Flow


Button ON
Send ‘0’
Send via Bluetooth
Button OFF
Send “1”
Set state of LED Widget
Python


Receive via Bluetooth
Receive ‘0’
Receive ‘1’
If else decision
if ‘0’
if ‘1’
LED ‘On’
LED ‘Off’
Arduino

## Arduino script

```
//Declare variable
char  serialData;

//Setup your baudrate & arduino pins
void setup()
{
   pinMode(13, OUTPUT);// LED pin on arduino
   Serial.begin(9600);
}

//Run your program again and again using loop
void loop()
{

 if (Serial.available() > 0) // send data only when you receive data from python
 {
     serialData = Serial.read();//read serial data and assign to variable
     Serial.print(serialData); //Print to check your data

     //Condition statement
     if(serialData == '0')
        {
         digitalWrite(13, HIGH); //Make your LED glow
        }
     else if(serialData == '1')
        {
          digitalWrite(13, LOW); //Make your LED off
        }

  }
}
```

***How it works***

This is similar to Chapter 7 except the communication of data is performed over Bluetooth. This arduino sketch assigns the pin 13 to be an output. This is where your LED is connected with. You can also make use of arduino's inbuilt LED to perform this operation. At the beginning of the loop, a serial read is triggered to read any data that comes in via serial port. Once the data is read, an ***if_else*** condition is set to change the LED's state. If the incoming data is '0', LED is set to HIGH and if incoming data is '1', the LED is set to LOW state.

***Verification***

Once you compile and upload this sketch to your arduino, then try to send '0' or '1' via arduino IDE's ***serial monitor***. This should make your LED to turn ON/ OFF accordingly. This is a basic sketch to begin with our exercises. Once you understand the logic of if_else statements to trigger an event, then its conven-

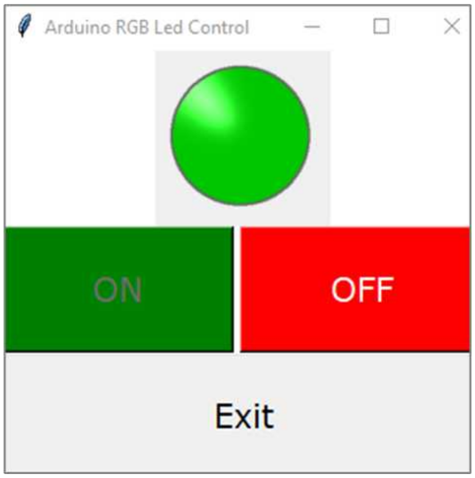
## Python script

The Python script remains same according to Chapter 7 exercise.

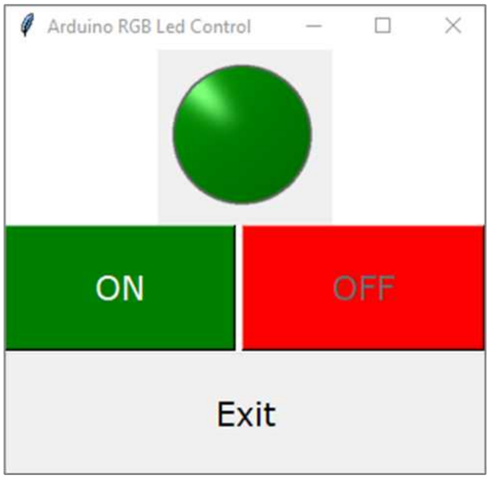
***Execution:*** Once the code is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- ♦ *Connect your hardware according to circuit diagram provided.*
- ♦ *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- ♦ *Verify the program using Arduino IDE Serial monitor.*
- ♦ *Once verified, Close the Serial monitor of Arduino IDE.*
- ♦ *Now run your Python script from Spyder IDE.*
- ♦ *Ta da! You will see your GUI opened.*

***Python GUI Output***



LED 'ON' State



LED 'OFF' State

## Python script

```
#Import following packages
import serial
import tkinter
import tk_tools

'''
Connect to HC-05 Bluetooth via serial port: Password: 1234
Goto --> Device manager to find the serail port number of BLE module
com7 --> Change port accordingly to yours as the bluetooth is connected
'''
arduino = serial.Serial('com6', 9600)

#Function call for led_on button
def led_on():
    arduino.write(b'0') #write this value to arduino
    led.to_green(True)  #Glow the led to green --> Refer to tk-tools docs
    button_on.config(state = "disabled") #disable button_on
    button_off.config(state = "normal") # enable button_off to normal state

#function call for led_off button
def led_off():
    arduino.write(b'1') #write this value to arduino
    led.to_green() #Switch off the led --> Refer to tk-tools docs
    button_off.config(state = "disabled") #disable button_off
    button_on.config(state = "normal") #enable button_on to normal state

#exit function call for closing the program
def close_window():
    arduino.close() #close Serial connection with arduino
    window.destroy() #destro tkinter app

# MAIN
window = tkinter.Tk() #create tkinter window
window.title("Arduino RGB Led Control") #give title
window.configure(background="white") #change background color
```

```
#Create buttons
button_on = tkinter.Button(window, text="ON",
                    font= ('Verdana',16), padx=50, pady =20,
                    bg="green",fg="white",
                    command = led_on)

button_off = tkinter.Button(window, text="OFF",
                    font=('Verdana',16), padx=50, pady =20,
                    bg="red",fg="white",
                    command = led_off)

button_exit = tkinter.Button(window, text="Exit",
                    font=('Verdana',16),
                    padx=130, pady =20,
                    command = close_window)

#Pack buttons
button_on.grid(row=1,column=1)
button_off.grid(row=1,column=2)
button_off.config(state = "disabled")
button_exit.grid(row=2,column=1, columnspan =2)

#Create interactive led using tk-tools
led = tk_tools.Led(window, size=100)
led.to_green() #set led to green switch off condition) --> Refer tk-tools docs
led.grid(row=0,column=1, columnspan =2)

#execute the loop
window.mainloop()
```

## CONCLUSION

In this chapter, we were able to control arduino wirelessly using a Bluetooth interface. This can be further expanded to WiFi and mobile network control based on our application needs. Usually this is applied in IOT sensor platform for data logging sensor’s.

# 12

## Challenge-8

# Fun with Pyfirmata

**In this Chapter, lets use Pyfirmata to control arduino via Python program. Instead of uploading for a unique program into arduino for each application, we will learn to use a Standard Firmata program for our microcontroller. Then we will learn to call each PINS of our micro-controller directly from python program.**

In this exercise you will get to learn to use python modules such as Pyfirmata tkinter, tkinter-tools to control the state of an LED connected to a micro-controller.

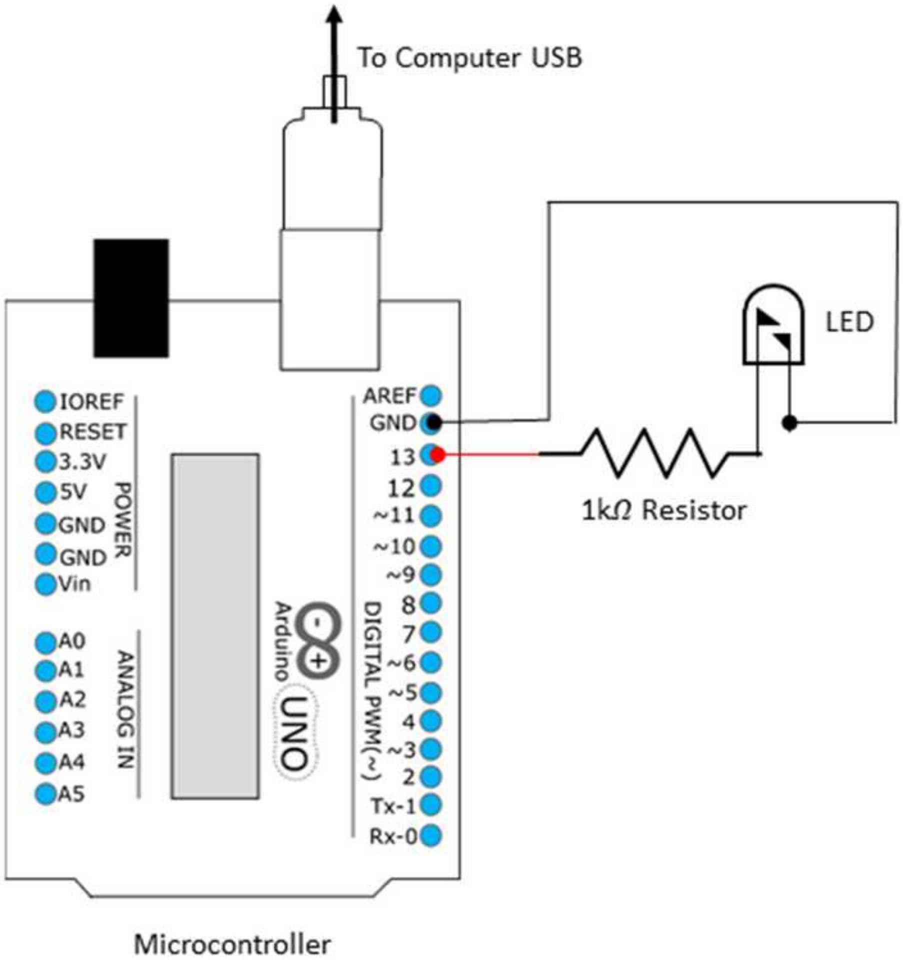
Refer to GitHub repository mentioned in Page 139 for the codes:

***ex8_pyfirmata_led_control.py***

## Hardware required

1. Arduino Uno – 1 Pc
2. LED – 1 Pc
3. Jumper wires (M to F)- 2 Pc
4. Bread Board—1 Pc (optional)

## Circuit schematic



You can also prefer to use inbuilt Arduino's LED at PIN 13

**Circuit connections explainer:**

LED's anode (longer leg) is connected to PIN 13 of arduino

LED's cathode (shorter leg) is connected to GND of arduino via a resistor

## Exercise

In the previous section, we uploaded a unique sketch to your Arduino board with respect to each exercises. Arduino sketches are written in a language similar to C++ and are compiled and recorded on the flash memory of the microcontroller when you press Upload. While you can use another language to directly program the Arduino microcontroller, it’s not a trivial task!

However, there are some approaches you can take to use Arduino with Python or other languages. One idea is to run the main program on a PC and use the serial connection to communicate with Arduino through the USB cable. The sketch would be responsible for reading the inputs, sending the information to the PC, and getting updates from the PC to update the Arduino outputs. In our previous exercise, we didn’t directly control the arduino from our PC. Instead, we were sending inputs to arduino or getting inputs from arduino via serial protocol.

To directly control Arduino from the PC, you’d have to design a protocol for the communication between the PC and Arduino. For example, you could consider a protocol with messages like the follow:

**VALUE OF PIN 13 IS HIGH:** used to tell the PC about the status of digital input pins

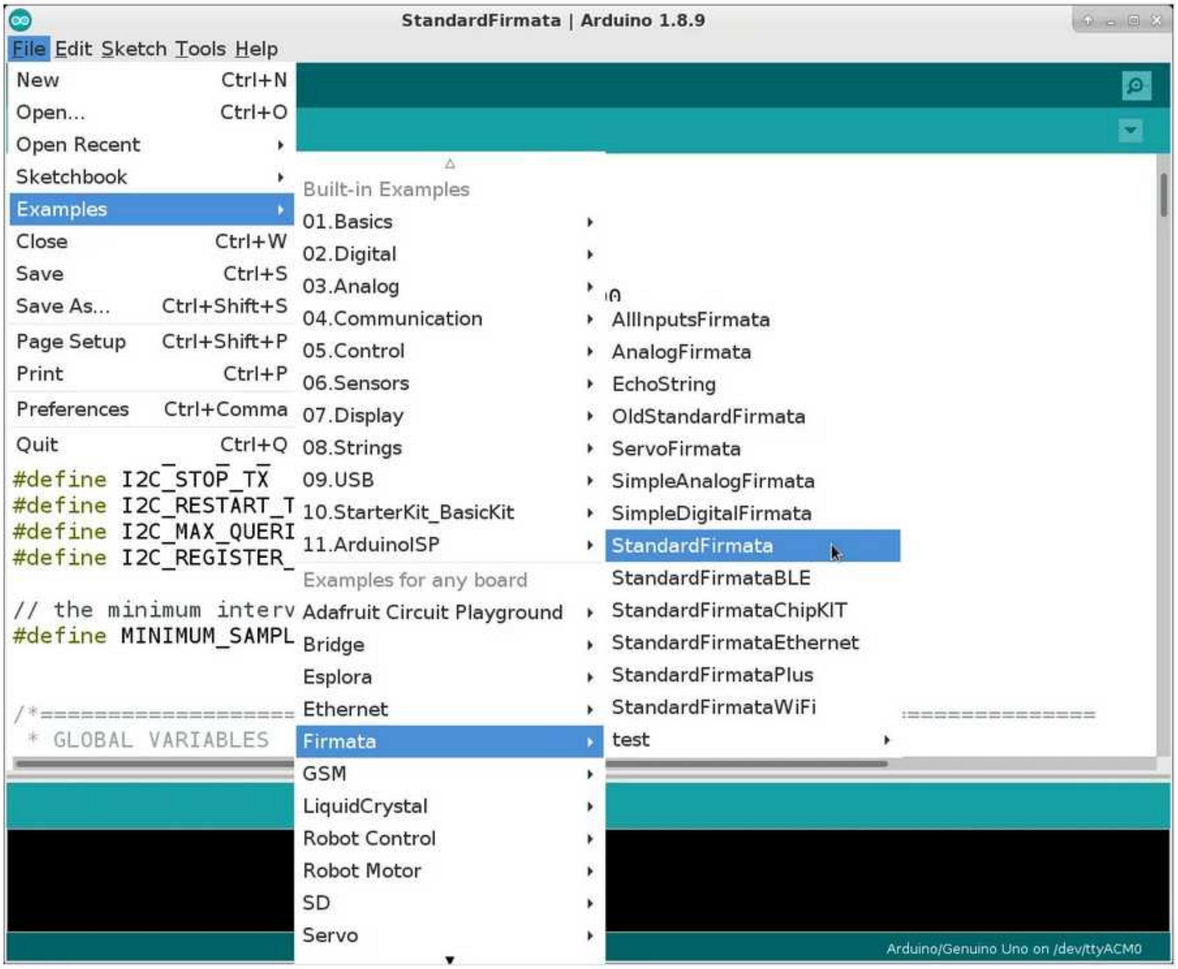
**SET PIN 11 LOW:** used to tell Arduino to set the states of the output pins

With the protocol defined, you could write an Arduino sketch to send messages to the PC and update the states of the pins according to the protocol. On the PC, you could write a program to control the Arduino through a serial connection, based on the protocol you’ve designed. For this, we can use Python and the ***Firmata*** library. This protocol establishes a serial communication format that allows you to read digital and analog inputs, as well as send information to digital and analog outputs.

The Arduino IDE includes ready-made sketches that will drive Arduino through Python with the Firmata protocol. On the PC side, there are implementations of the protocol in several languages, including Python. To get started with Firmata, let’s use it to implement a “Hello, World!” program

### Uploading the Firmata Sketch

Before we write your Python program to drive Arduino, you have to upload the Firmata sketch so that you can use that protocol to control the board. The sketch is available in the Arduino IDE’s built-in examples. To open it, access the *File* menu, then *Examples*, followed by *Firmata*, and finally *StandardFirmata*:



```
#define I2C_STOP_TX
#define I2C_RESTART_T
#define I2C_MAX_QUERI
#define I2C_REGISTER_

// the minimum interv
#define MINIMUM_SAMPL


/*====================
 * GLOBAL VARIABLES
```

The sketch will be loaded into a new IDE window. To upload it to the Arduino, you can follow the same steps you did before:

1. Plug the USB cable into the PC.
2. Select the appropriate board and port on the IDE.

After the upload is finished, you won’t notice any activity on the Arduino. To control it, you still need a program that can communicate with the board through the serial connection. To work

**for installation in pip and windows, run:**

```
>> pip install pyfirmata
```

**Usage:**

```
>> import pyfirmata
```

## Sample Python script to try

```
import pyfirmata
import time

board = pyfirmata.Arduino('/dev/ttyACM0')

while True:
    board.digital[13].write(1)
    time.sleep(1)
    board.digital[13].write(0)
    time.sleep(1)
```

***How it works***

Here’s how this program works. You import pyfirmata and use it to establish a serial connection with the Arduino board, which is represented by the board object in line 4. You also configure the port in this line by passing an argument to pyfirmata.Arduino(). You can use the Arduino IDE to find the port.

board.digital is a list whose elements represent the digital pins of the Arduino. These elements have the methods read() and write(), which will read and write the state of the pins. Like most embedded device programs, this program mainly consists of an infinite loop:

In line 7, digital pin 13 is turned on, which turns the LED on for one second.

In line 9, this pin is turned off, which turns the LED off for one second.

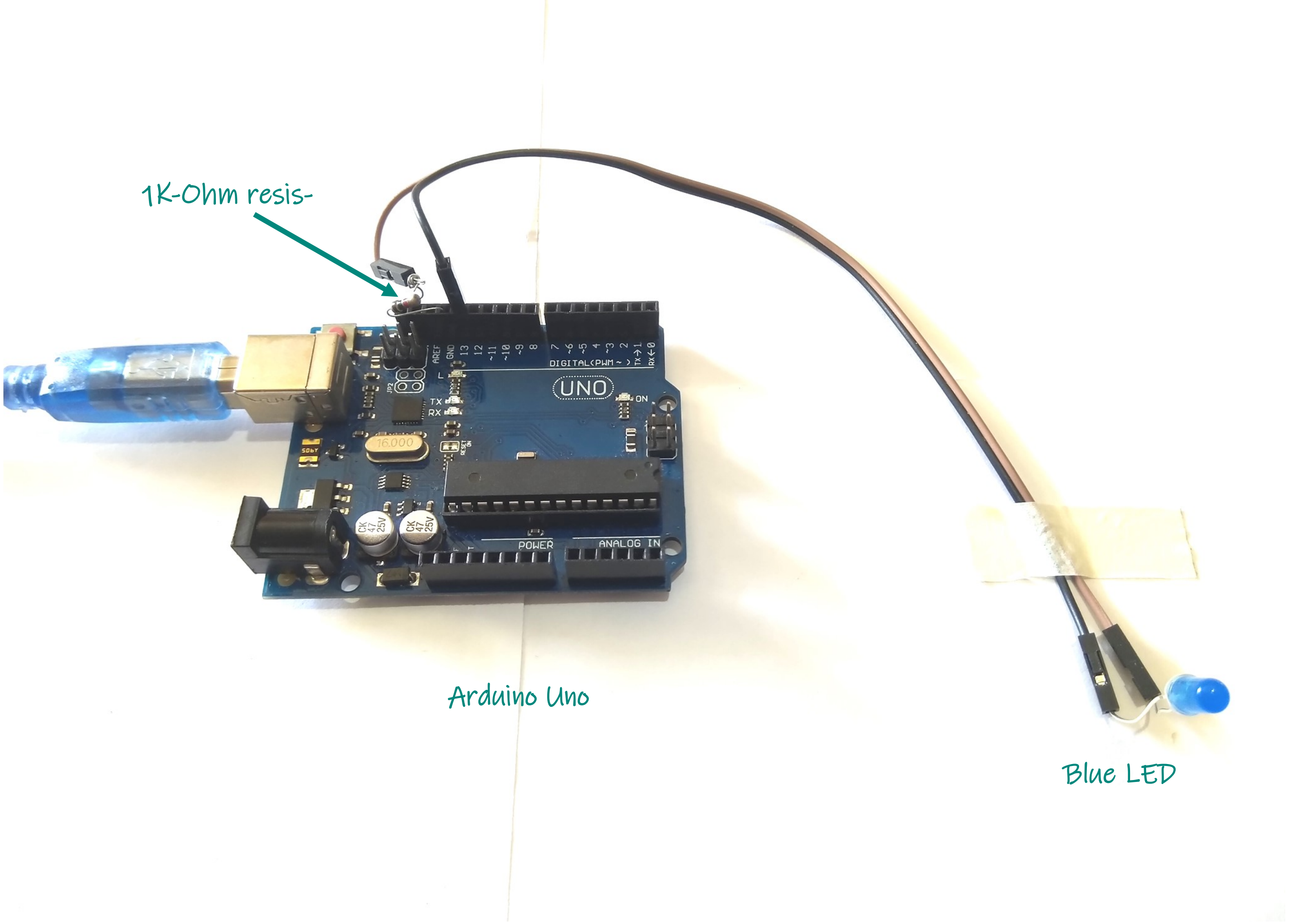
## Python script with tkinter

```
#Import following packages
import pyfirmata
import time
import tkinter
import tk_tools

board = pyfirmata.Arduino('com6')

#Function call for led_on button
def led_on():
    board.digital[13].write(1)  #write this value to arduino's 13th PIN'
    led.to_green(True)  #Glow the led to green --> Refer to tk-tools docs
    button_on.config(state = "disabled") #disable button_on
    button_off.config(state = "normal") # enable button_off to normal state
    time.sleep(1)
```

```
#function call for led_off button
def led_off():
    board.digital[13].write(0) #write this value to arduino
    led.to_green() #Switch off the led --> Refer to tk-tools docs
    button_off.config(state = "disabled") #disable button_off
    button_on.config(state = "normal") #enable button_on to normal state
    time.sleep(1)

def close_window():
    window.destroy() #destro tkinter app

window = tkinter.Tk() #create tkinter window
window.title("Arduino RGB Led Control") #give title
window.configure(background="white") #change background color

button_on = tkinter.Button(window, text="ON",
                           font= ('Verdana',16), padx=50, pady =20,
                           bg="green",fg="white",
                           command = led_on)

button_off = tkinter.Button(window, text="OFF",
                            font=('Verdana',16), padx=50, pady =20,
                            bg="red",fg="white",
                            command = led_off)

button_exit = tkinter.Button(window, text="Exit",
                             font=('Verdana',16),
                             padx=130, pady =20,
                             command = close_window)

button_on.grid(row=1,column=1)
button_off.grid(row=1,column=2)
button_off.config(state = "disabled")
button_exit.grid(row=2,column=1, columnspan =2)
led = tk_tools.Led(window, size=100)
led.to_green() #set led to green switch off condition) --> Refer tk-tools docs
led.grid(row=0,column=1, columnspan =2)

window.mainloop()
```

- *Connect your hardware according to circuit diagram provided.*
- *Upload your Firmata arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! Control the LED using your GUI and real time control of arduino.*

## Hardware setup



You will get the same output as Challenge-1 with GUI buttons to control ON/OFF the LED on your arduino / external LED connected to PIN 13.

### CONCLUSION

In this chapter, we experimented an alternate way of hardware interfacing with Arduino from Python using Firmata library. This is very useful as we don’t have to program the arduino each and every time when we update our Python program.

# 13 Capstone Project

# BUILD & CONTROL OF A ROBOT VEHICLE

**In this exercise, you will learn to build a 4-wheeled robot from scratch, program it and control it in wireless protocol using Bluetooth with a help of tkinter GUI developed over python.**

In this exercise you will get to learn to use python modules such as pyserial, tkinter, tkinter-tools to control the speed and direction of 4-wheeled robotic vehicle connected to a micro-controller. You will learn to create slider widgets, dial gauge widget, arrange buttons on tkinter window, send a speed and direction data to micro-controller based on the slider position and button event calls.

Refer to GitHub repository mentioned in Page 139 for the codes:

***ex8_4wheel_car_control_bluetooth.py***

***e8_4wheel_car_control_bluetooth.ino***

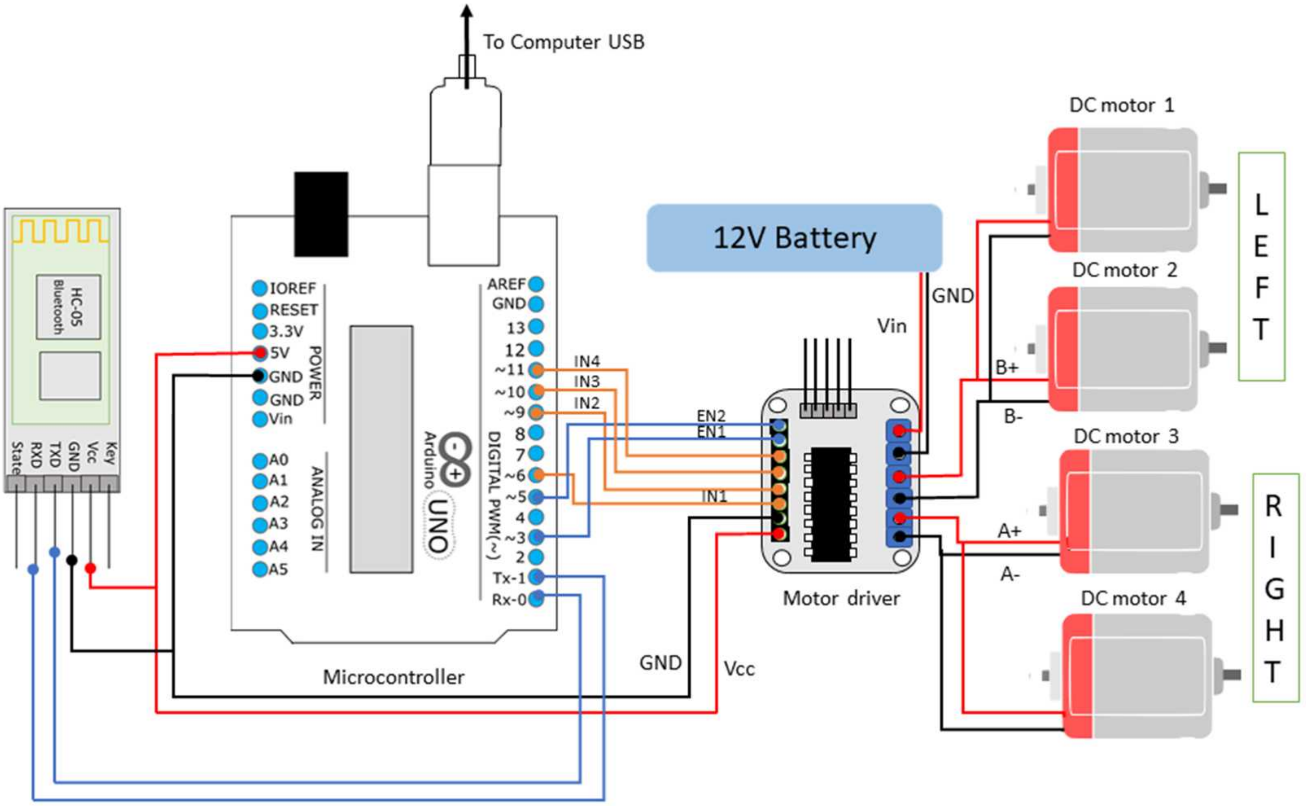
## Exercise

This exercise is about to build, program and wireless control of 4-Wheel robotic vehicle connected to arduino micro-controller using a Tkinter GUI built with Python script.

## Hardware required

1. Arduino Uno – 1 Pc
2. 7.4V/9V Battery – 1 Pc
3. Jumper wires (F to M)- Few Pcs
4. Bread Board—1 Pc (optional)
5. DC Motors—4 PCs
6. Wheels - 4 PCs
7. Motor driver—1 PC (Any motor driver of your choice)
8. Bluetooth (HC-05) - 1 Pc
9. Masking tape
10. Card board for making chassis (You can use any chassis if you have already)
11. Glue gun (For fixing motors to car board)

## Circuit schematic



## Let’s Build

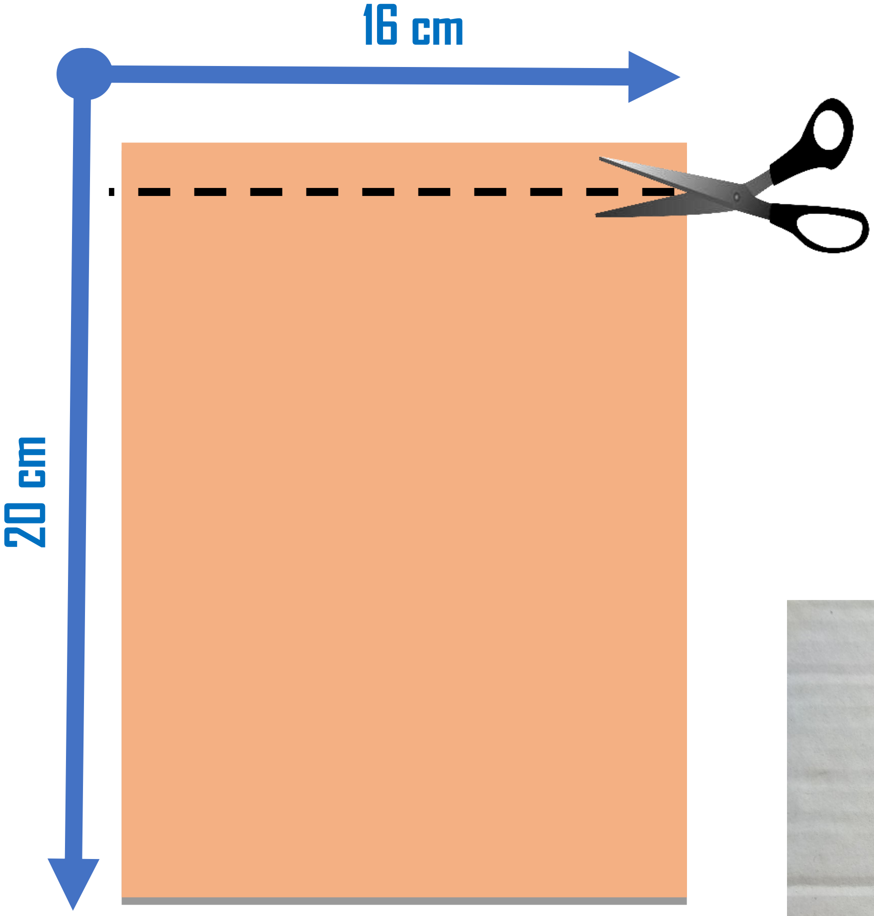
This build tutorial is based on ***Scrap-o-bot concept***. So, we will make use of scraps around our house like carb board, masking tapes etc. .

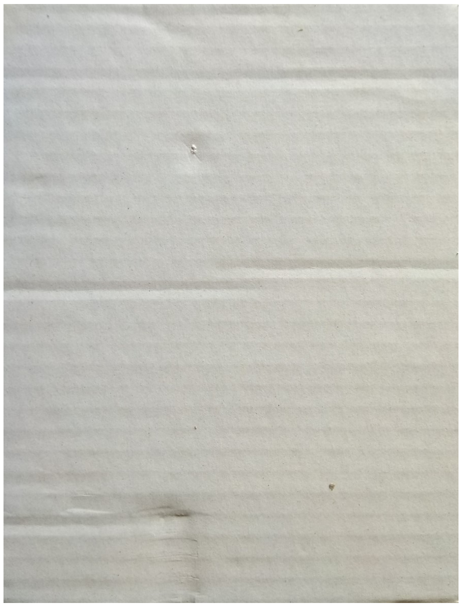
01

Let’s make **CHASSIS!**

Chassis forms the base of our robotic vehicle.

To begin with, take a piece of Cardboard and Cut them according to dimensions given below. Use scissors/ cutter to achieve this step. Take the help of elderly person to handle scissors.

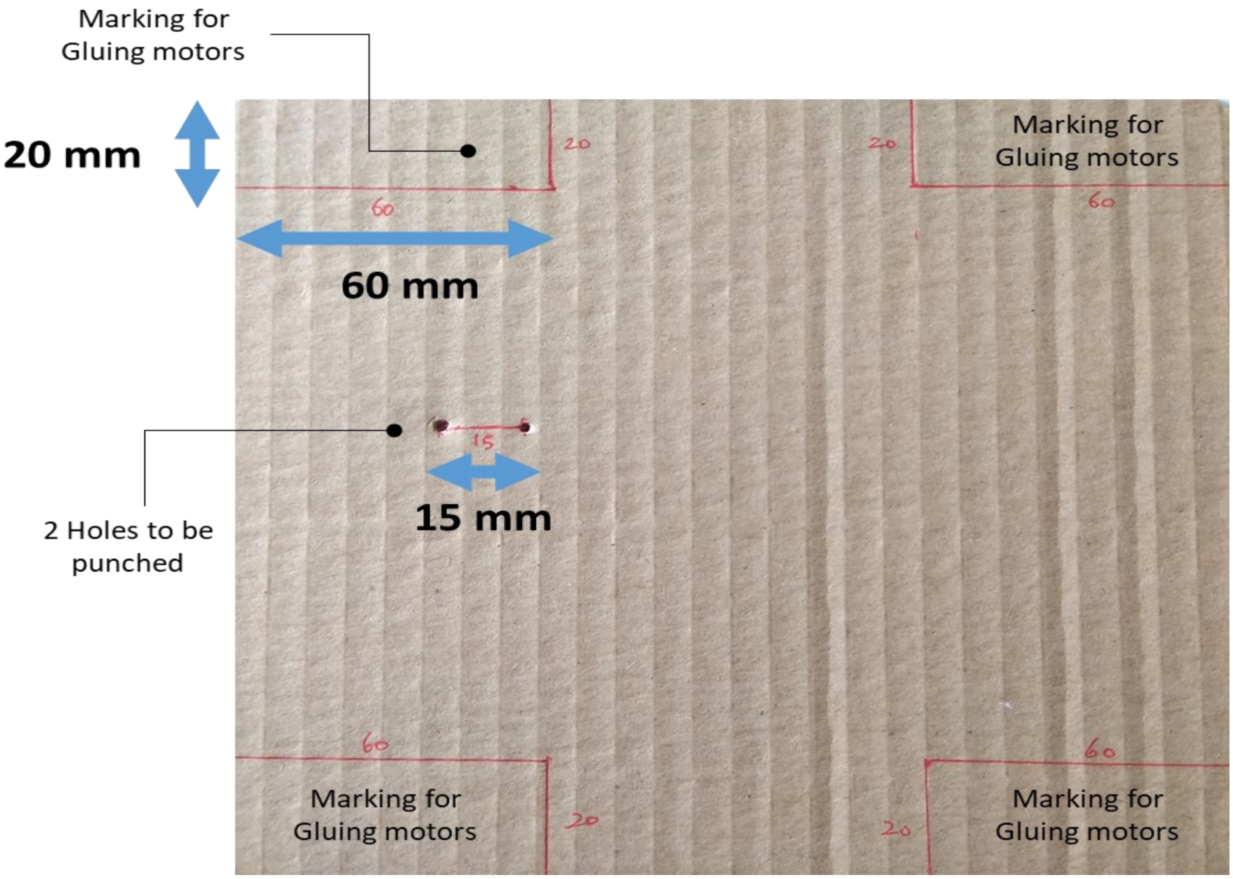




Card board

## 02

Make a hole from bottom for letting Motor wires to pass through it. You can make use of screw driver / Scissors to make a hole. Then mark dimensions with a marker pen/pencil for gluing your motors to chassis as shown below. This will act as a reference for fixing your motors in next step.

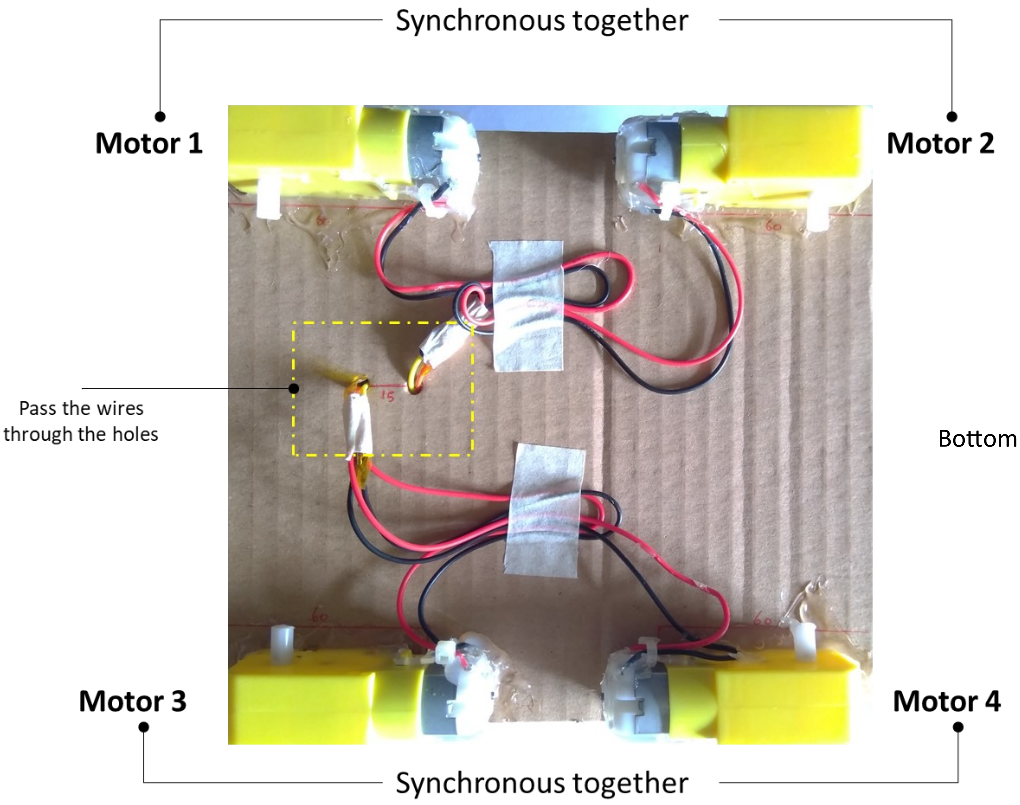

Marking for Gluing motors
20 mm
60 mm
20
20
60
60
Marking for Gluing motors
2 Holes to be punched
15
15 mm
60
60
Marking for Gluing motors
20
20
Marking for Gluing motors

# 03

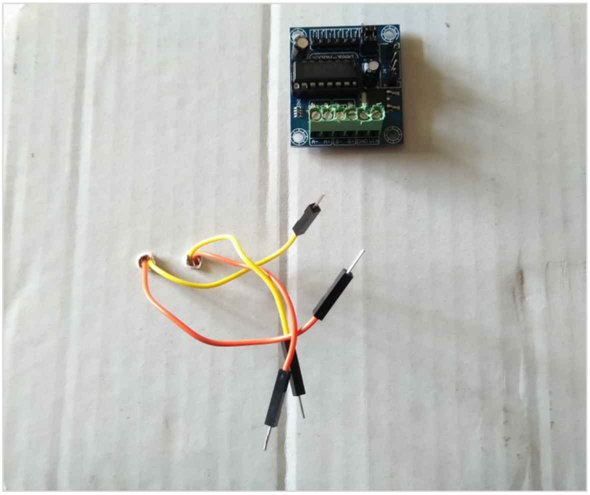
The next step is to glue your Geared DC Motors (Any motor of your wish) to the chassis with a help of Hot glue gun / Hot melt.

Align the wires of each motor and use a electrical tape or masking tape to fix them in place. This will result you with tidy circuitry.

Refer to motor connection as shown in Circuit diagram. The 2 motors on the left side are synchronous together and so connected together. This applies to set f right side motors.



Bottom View

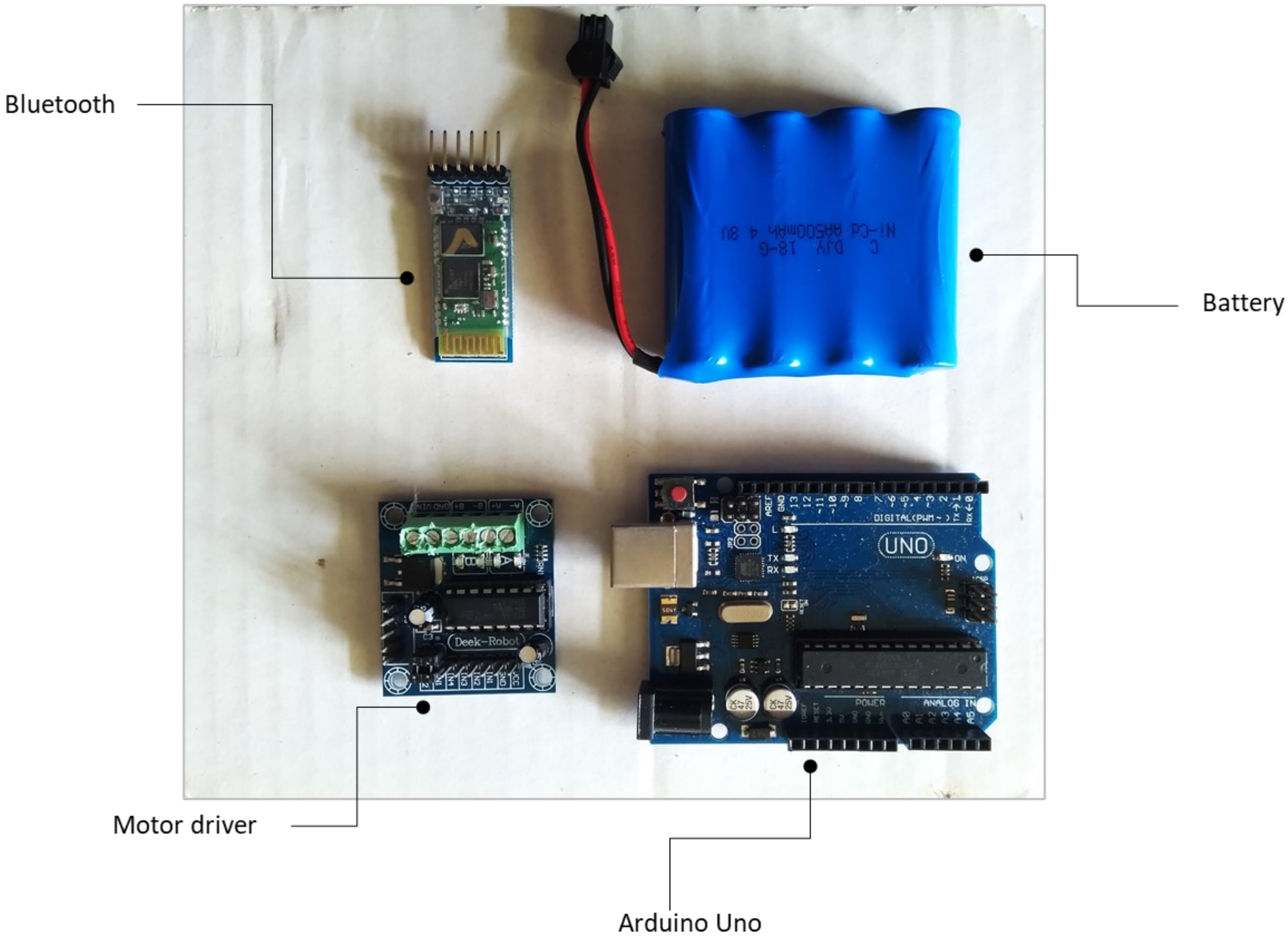


Top View

# 04

## Mount the **electronics!**

As a next step, lets mount our Arduino, Motor driver, Battery and Bluetooth on our chassis. By mounting, it means we can fix them using masking tape or using Hot glue gun melt.

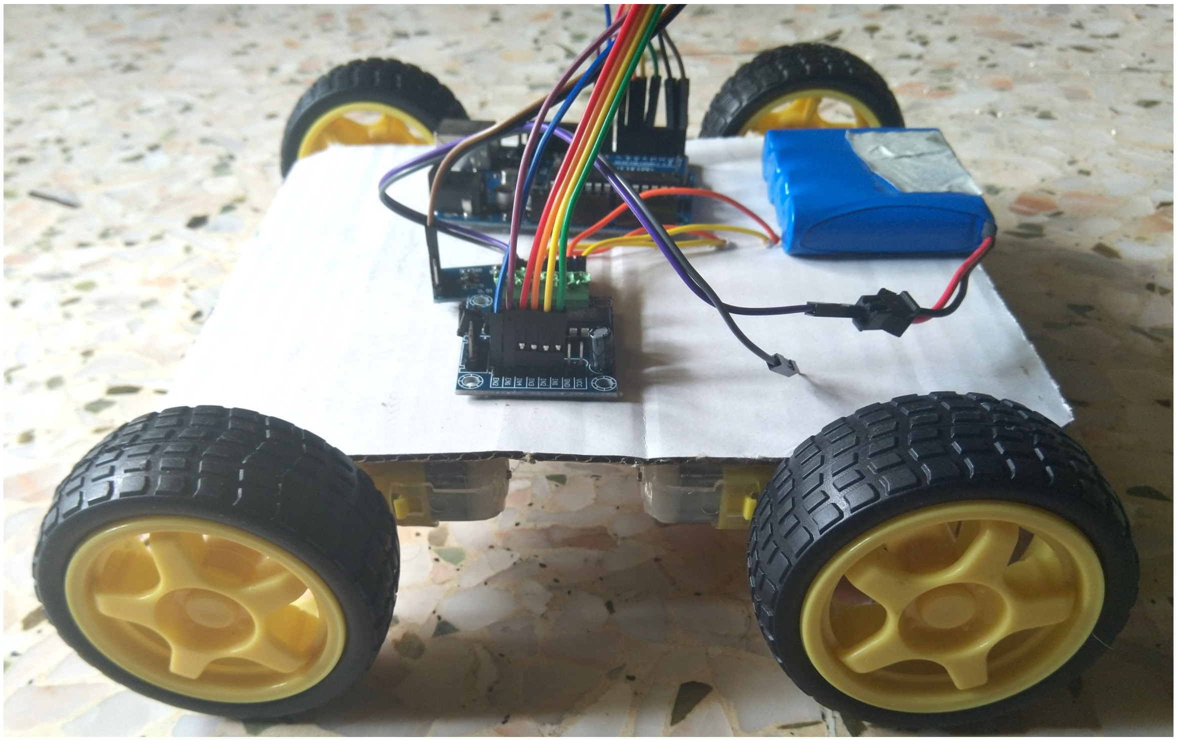


05

Mount the **Wheels and Hook up electronics!**

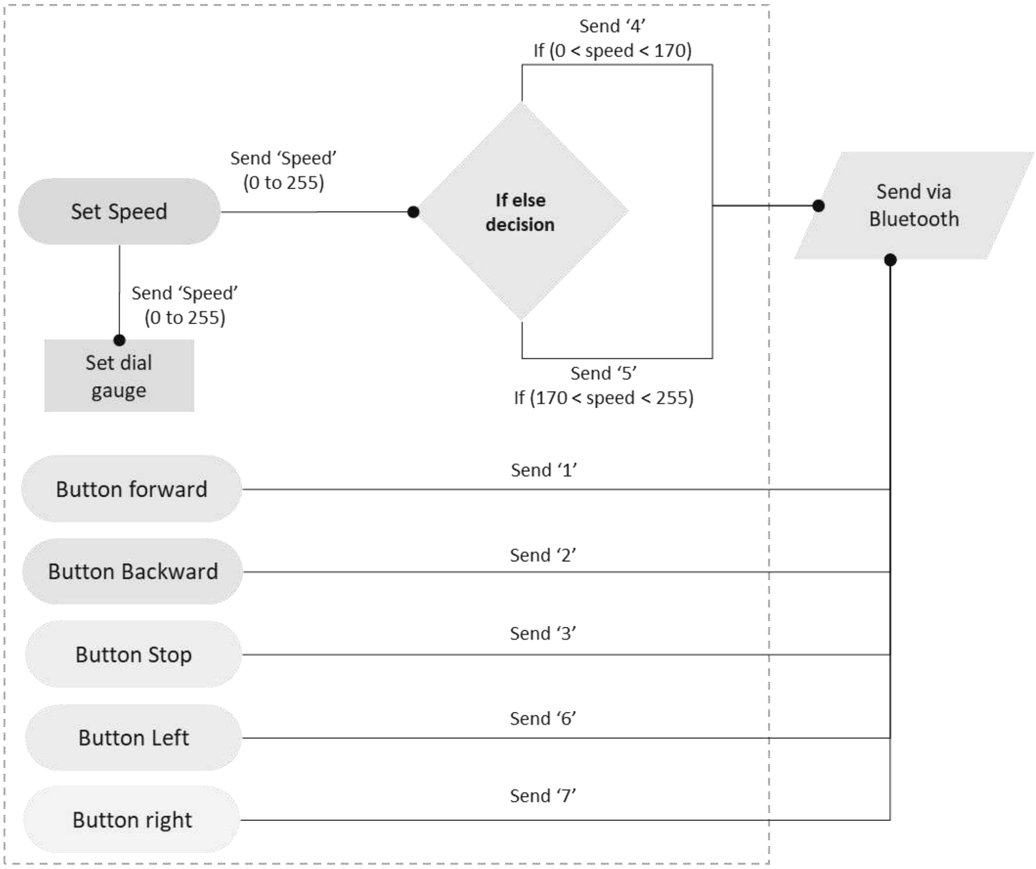
Next, mount the wheels that fits your motor and hook up the electronics according to circuit diagram shown previously.

Before connecting the arduino to Bluetooth, Please make sure to upload the arduino sketch.

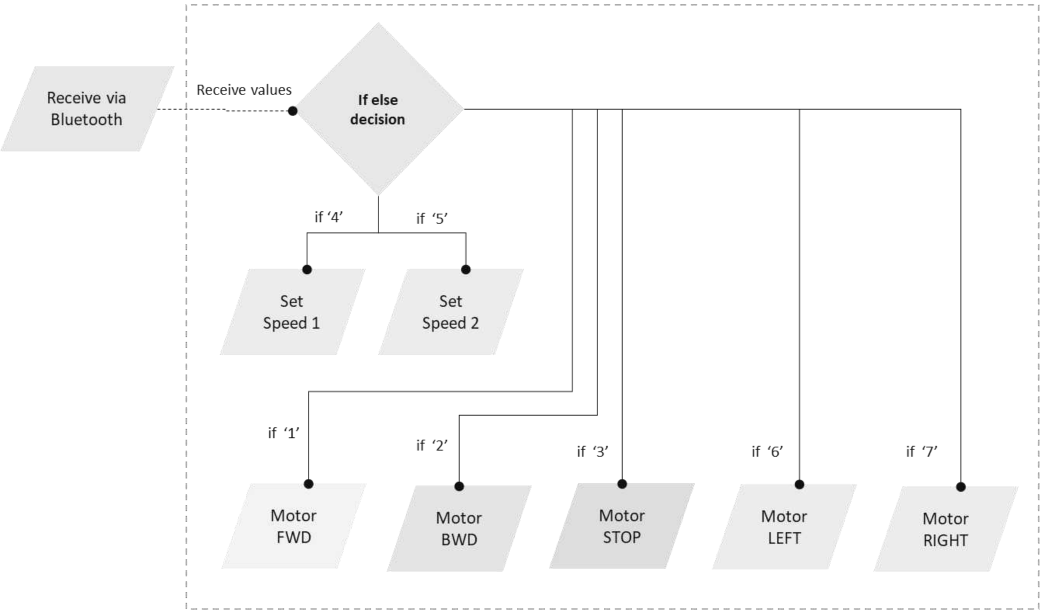


Final look of a robot vehicle

## Work Flow



Python



Arduino

## Arduino script

```
//Declare variables
char  serialData;

//Declaring Pins of arduino to pins of Deek robot 2 Channel motor driver (L293D)
//Play around your driver to control the direction of motors
int m1=6; //connects to IN2 pin of Deek Robot motor driver -->Sets the direction
int m2=9; //Connects to IN1 pin of Deek Robot motor driver --> Sets the direction
int e1=5; //Connects to EN1 pin of Deek Robot motor driver -->Sets the speed

int m3=10; //connects to IN3 pin of Deek Robot motor driver -->Sets the direction
int m4=11; //Connects to IN4 pin of Deek Robot motor driver -->Sets the direction
int e2=3; //Connects to EN2 pin of Deek Robot motor driver -->Sets the speed

int x =0; //Variable to store the speed value from Python

//Setup your baudrate & arduino pins
void setup()
{
        pinMode(m1, OUTPUT);
        pinMode(m2, OUTPUT);
        pinMode(m3, OUTPUT);
        pinMode(m4, OUTPUT);
        pinMode(e1, OUTPUT);
        pinMode(e2, OUTPUT);
        Serial.begin(9600);
}
void loop()
{
     // send data only when you receive data from python
    if (Serial.available() > 0)
    {
      serialData = Serial.read();
      Serial.print(serialData);
```

```
//Forward motion of car
    if(serialData == '1')
    {
      //Motor set 1
      analogWrite(e1, x);
      digitalWrite(m1, 1);
      digitalWrite(m2, 0);

      //Motor set 2
      analogWrite(e2, x);
      digitalWrite(m4, 1);
      digitalWrite(m3, 0);
     }

    //Reverse motion of car
     else if (serialData == '2')
     {
       //Motor set 1
      analogWrite(e1, x);
      digitalWrite(m1, 0);
      digitalWrite(m2, 1);

      //Motor set 2
      analogWrite(e2, x);
      digitalWrite(m4, 0);
      digitalWrite(m3, 1);
     }

     //Apply brakes
     else if (serialData == '3')
     {
       //Motor set 1
       digitalWrite(m1, 0);
       digitalWrite(m2, 0);

       //Motor set 2
       digitalWrite(m3, 0);
       digitalWrite(m4, 0);
     }
```

```
//set Speed level 1
        else if (serialData == '4')
        { x=80;
        }

        //set Speed level 2
        else if (serialData == '5')
        {  x=150;
        }
        //set Speed level 3
        else if (serialData == '6')
        { x=255;
        }
        //Turn left
        else if (serialData == '7')
        {
          //Motor set 1
         analogWrite(e1, x);
         digitalWrite(m1, 1);
         digitalWrite(m2, 0);
         //Motor set 2
         analogWrite(e2, x);
         digitalWrite(m4, 0);
         digitalWrite(m3, 1);
        }
        //Turn right
        else if (serialData == '8')
        {
          //Motor set 1
         analogWrite(e1, x);
         digitalWrite(m1, 0);
         digitalWrite(m2, 1);
         //Motor set 2
         analogWrite(e2, x);
         digitalWrite(m4, 1);
         digitalWrite(m3, 0);
        }
    }
}
```

***How it works***

This is similar to Chapter 9 except the communication of data is performed over Bluetooth. Additional functions are included for robot vehicle navigation such as going forward, backward, left, right and stop. The number of speed levels remains the same. And the code functionality remains the same with if_else statements.

***Verification***

Once you compile and upload this sketch to your arduino, then try to send '0' or '1' via arduino IDE's ***serial monitor***. This should make your motor to function accordingly. Try all the navigation options by sending numbers from 1 to 6 via serial monitor. You can use the arduino serial monitor to trouble shoot your motor connections from your motor driver to arduino micro controller. You can make changes in code or you can make changes to electronics wiring.

## Python script

The Python script remains almost same but with additional functions according to Chapter 9 exercise.  We have added new functions for Moving forward, backward of the robotic vehicle.

***Execution:*** Once the code is combined together, we are ready to execute our python. But before doing this, make sure you have completed the following action check list as done in previous exercise.

- *Connect your hardware according to circuit diagram provided.*
- *Upload your arduino sketch to your arduino microcontroller via USB cable connected to your PC.*
- *Verify the program using Arduino IDE Serial monitor.*
- *Once verified, Close the Serial monitor of Arduino IDE.*
- *Now run your Python script from Spyder IDE.*
- *Ta da! You will see your GUI opened.*

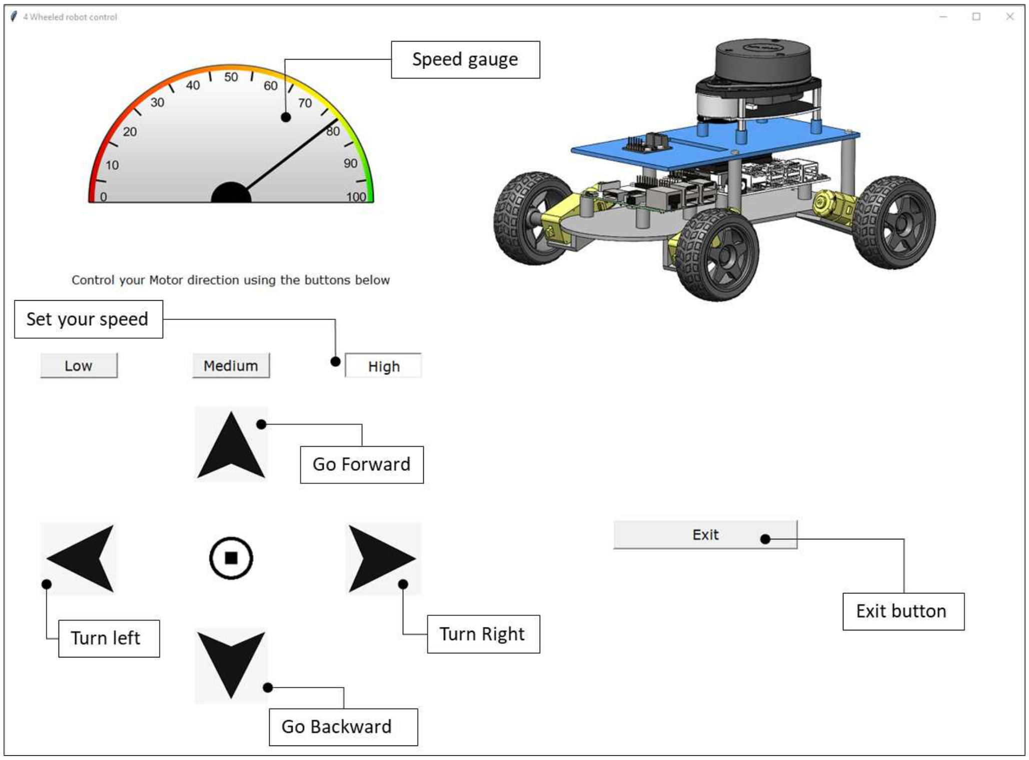
## Python script

```
#Import following packages
import serial
import tkinter
import tkinter.messagebox
from PIL import Image, ImageTk

'''
Connect to HC-05 Bluetooth via serial port: Password: 1234
Goto --> Device manager to find the serail port number of BLE module
com7 --> Change port accordingly to yours as the bluetooth is connected
'''
arduino = serial.Serial('com6',9600)

#Function call to close your program
def close_window():
    arduino.close()
    window.destroy()

#function call to set speed of motor
def speed1():
    speed_scale.config(image = speed_low) #set image to button
    arduino.write(b'4') #write to arduino

def speed2():
    speed_scale.config(image = speed_medium) #set image to button
    arduino.write(b'5') #write to arduino

def speed3():
    speed_scale.config(image = speed_high) #set image to button
    arduino.write(b'6') #write to arduino
#function call to move your car forward
def upp():
    arduino.write(b'1')

#function call to move your car backward
def downn():
    arduino.write(b'2')
```

```
#function call to turn your car left
def left():
    arduino.write(b'7')

#function call to turn your car right
def right():
    arduino.write(b'8')

#function call to stop your car
def stop():
    arduino.write(b'3')

# MAIN
window = tkinter.Tk() #create tkinter window
window.title("4 Wheeled robot control") #give title
window.configure(background="white") #change background color
window.geometry("1400x1000") #set size of tkinter window

#create frame to stack your control buttons
f1 = tkinter.Frame(window)
f1.grid(row=2, column=0, sticky="nsew", pady=40)
f1.configure(background="white")

#Declare the images
stopped = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/stop.png"))
up = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/up.png"))
down = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/down.png"))
left_turn = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/left_turn.png"))
right_turn = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/right_turn.png"))
car = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/car.png"))
speed_low = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/exercises/speed1.png"))
speed_medium = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/exercises/speed2.png"))
speed_high = ImageTk.PhotoImage(Image.open("C:/Users/User/Documents/scripts/exercises/speed3.png"))
```

```
#Create your widgets
low_button = tkinter.Radiobutton(f1, text="Low",
                        indicatoron=False, value="low", width=8,font=('Verdana',14),
command = speed1)


medium_button = tkinter.Radiobutton(f1, text="Medium",
                        indicatoron=False, value="medium", width=8, font=
('Verdana',14), command = speed2)


high_button = tkinter.Radiobutton(f1, text="High",
                        indicatoron=False, value="high", width=8, font=
('Verdana',14), command = speed3)


l1 = tkinter.Label(window, text="Control your Motor speed & direction using the
buttons below",
                font=('Verdana',12), bg="white", fg="black")


button_exit = tkinter.Button(window, text="Exit", font = ('Verdana',14),
            padx=100, command=close_window)


buttonup = tkinter.Button(f1, image = up, bg = "white", command= upp,
                       relief= tkinter.FLAT )


buttonstop = tkinter.Button(f1, image = stopped, bg = "white", command= stop,
                        relief= tkinter.FLAT)


buttondown = tkinter.Button(f1, image = down, bg ="white", command= downn,
                        relief= tkinter.FLAT)


buttonleft = tkinter.Button(f1, image = left_turn, bg ="white", command= left,
                 relief= tkinter.FLAT)


buttonright = tkinter.Button(f1, image = right_turn, bg ="white", command=
right,
                 relief= tkinter.FLAT)


#fun image for your car
#you can replcae with your car image
car_image = tkinter.Button(window, image = car, bg ="white",
                relief= tkinter.FLAT)
```

```
speed_scale = tkinter.Button(window, image = speed_low, bg ="white", padx
=100, pady=100, relief= tkinter.FLAT)



#pack all your widgets
low_button.grid(row=2, column=0, pady=5)
medium_button.grid(row=2, column=1, pady=5)
high_button.grid(row=2, column=2, pady=5)
l1.grid(row=1, column=0,  columnspan =3)
button_exit.grid(row=2,column=3)
buttonup.grid(row =3, column =1,pady =30, padx=50)
buttonstop.grid(row =4, column =1, pady =20, padx=50)
buttondown.grid(row =5, column =1, padx=50, pady =20)
buttonleft.grid(row =4, column =0, padx=50, pady =20)
buttonright.grid(row =4, column =2, padx=50, pady =20)
car_image.grid(row =0, column =3, rowspan = 2)
speed_scale.grid(row =0, column =0)

#execute the loop
window.mainloop()
```

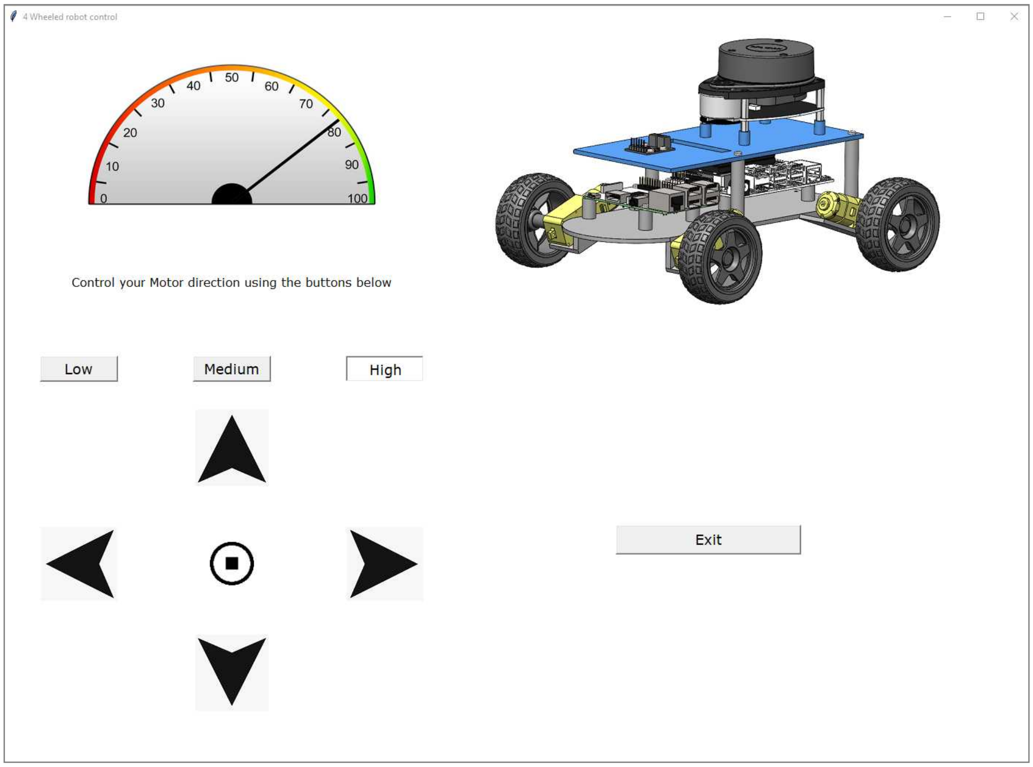


4 Wheeled robot control

Control your Motor direction using the buttons below

GUI explained

***Python GUI Output***



4 Wheeled robot control

Control your Motor direction using the buttons below

Active GUI

## POTENTIAL APPLICATIONS

- Self driving delivery vehicle control and monitoring
- Robot based surveillance system

Self driving robotic vehicle

## CONCLUSION

Based on this capstone project, we were able to build a robotic vehicle from scratch, combine the electronics together and create a python tkinter program to develop a GUI with interfaces to control our robotic vehicle. This basic robotic systems can be further developed to solve the real world problems.

# CONCLUSION

From this book, we hope you got introduced to Arduino and Python programming. We discussed the procedure to install Arduino and Python IDE on your PC. We had an overview to Tkinter modules and various elements available for building an effective Graphical User interface from scratch. We also got introduced to Tkinter high level widgets and the ways to implement it in our project. Various challenges and the final capstone project that you came across are designed carefully to give you an idea about the possibilities of GUI creation for your mechatronics/ robotics projects. “Learning by doing is a best way to learn new things” so try to implement your learnings in your upcoming projects to get more familiar in programming electronics.

***We wish you good luck for future projects!***

***Keep learning!***

# SOURCE CODE

The Source code for the exercises in this book is available in **GitHub repository.**

**https://github.com/robots-guy/Arduino_Python_programming_for_Robots**

Or alternatively you can **Click me!**

*Introduction to UI based computer control*

# ARDUINO + PYTHON PROGRAMMING FOR ROBOTS

## POWERED BY PYTHON 3