CHAPTER 5

DESIGN GRAPHICAL USER INTERFACE AND REGULAR EXPRESSION

Chapter Outline

5.1 DESIGN A GRAPHICAL INTERFACE USING TKINTERLIBRARY

Graphical User Interface (GUI) is nothing but a desktop application which helps you to interact with the computers. They are used to perform different tasks in the desktops, laptops and other electronic devices.

- GUI apps like Text-Editors are used to create, read, update and delete different types of files.
- GUI apps like Sudoku, Chess and Solitaire are games which you can play.
- GUI apps like Google Chrome, Firefox and Microsoft Edge are used to browse through the Internet.
- Creating a Calculator which would have a user-interface and functionalities that persists in a calculator.
- IDE's for coding are on a GUI app.

They are some different types of GUI apps which we daily use on the laptops or desktops. We are going to learn how to create those types of apps.

Python provides various options for developing Graphical User Interfaces (GUIs). Most important are listed below :

1. **Tkinter :** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. It is easiest to start with. Tkinter is Python's standard GUI (graphical user interface) package. It is the most commonly used toolkit for GUI programming in Python.

   Note : In tkinter module first letter T is CAPITAL LETTER in python 2.x and t is SMALL LETTER in 3.x version.

2. **wxPython :** This is an open-source, cross-platform GUI toolkit written in C++. It is one of the alternatives to Tkinter, which is bundled with Python. http://wxpython.org.
3. **JPython :** JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine. http://www.jython.org.
4. **Kivy :** Kivy is a free and open source Python library for developing mobile apps and other multitouch application software with a natural user interface. It is distributed under the terms of the MIT License, and can run on Android, iOS, GNU/Linux, OS X, and Windows. https://kivy.org/#home.
5. **Python QT :** PyQt is a GUI widgets toolkit. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt is a blend of Python programming language and the Qt library. This introductory tutorial will assist you in creating graphical applications with the help of PyQt. https://www.qt.io/qt-for-python

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Introduction to tkinter Package : Tkinter is actually an inbuilt Python module used to create simple GUI apps. It is the most commonly used module for GUI apps in the Python. You don't need to worry about installation of the Tkinter module as it comes with Python default. Tkinter is the Python port for ***Tcl-Tk GUI*** toolkit developed by Fredrik Lundh. This module is bundled with standard distributions of Python for all platforms.

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- **tkinter :** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.
- Tkinter is distributed along with Python software.
- It is a platform independent package.
- It has variety of GUI elements such as Label, Button, Menu, Frame..etc.,
- These GUI elements are called ***"Widgets"***.

Tkinter programming : Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps :

1. Import the Tkinter module.
2. Create the GUI application main window.
3. Add one (or) more of the above-mentioned widgets to the GUI application.
4. Enter the main event loop to take action against each event triggered by the user.

Consider the following diagram, it shows how an application actually executes in Tkinter :

To start out with, we first import the Tkinter model. Followed by that, we create the main window. It is in this window that we are performing operations and displaying visuals and everything basically. Later, we add the widgets and lastly we enter the main event loop.

If you noticed, there are 2 keywords here that you might not know at this point. These are the 2 keywords :

- Widgets.
- Main Event Loop.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

An event loop is basically telling the code to keep displaying the window until we manually close it. It runs in an infinite loop in the back-end.

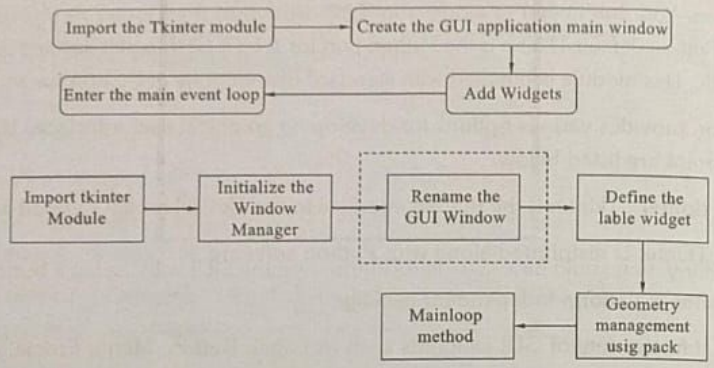


FIG 5.1 :

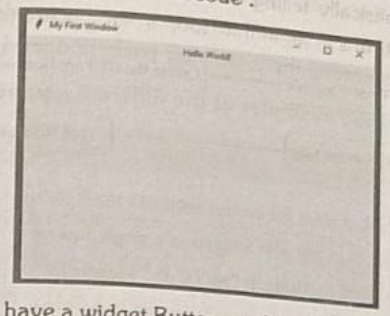
EXAMPLE

```
import tkinter
window = tkinter.Tk()
window.title("My First Window") # to rename the title of the window
# Code to add widgets will go here...
label = tkinter.Label(window, text = "Hello World!").pack() # pack is used to show the object in the window
window.mainloop()
```

- First, you import the key component, i.e., the tkinter module.
- As a next step, you initialize the window manager with the tkinter.Tk() method and assign it to a variable. This method creates a blank window with close, maximize and minimize buttons on the top as a usual GUI should have.
- Then as an optional step, you will rename the title of the window as you like with window.title(title_of_the_window).
- Next, you make use of a widget called ***Label,*** which is used to insert some text into the window.
- Then, you make use of Tkinter's geometry management attribute called **pack()** to display the widget in size it requires.
- Finally, as the last step, you use the mainloop() method to display the window until you manually close it. It runs an infinite loop in the backend.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

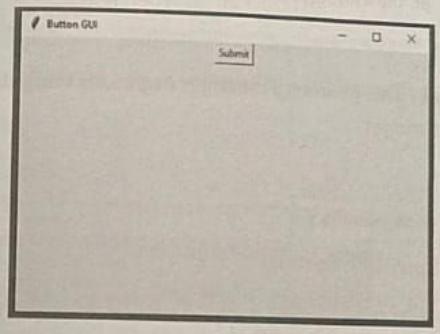
Check out the output for the above code :



Similarly, you could have a widget Button, and the GUI will display a button instead of some text (Label).

```
import tkinter
window = tkinter.Tk()
window.title("Button GUI")
button_widget = tkinter.Button(window,text="Submit")
button_widget.pack()
tkinter.mainloop()
```

Check out the output for the above code :



5.2 DESIGN GUI USING DIFFERENT GEOMETRY MANAGERS

Just instantiating (creating) a widget does not necessarily mean that it will appear on the screen. To get the widget to appear, we need to tell the parent widget where to put it. To do that, we use one of Tkinter's three geometry managers (also known as ***layout managers***). A geometry manager is some code that runs on the backend of Tkinter (we don't interact with the geometry managers directly). We simply choose which geometry manager we want to use and give it some parameters to work with.

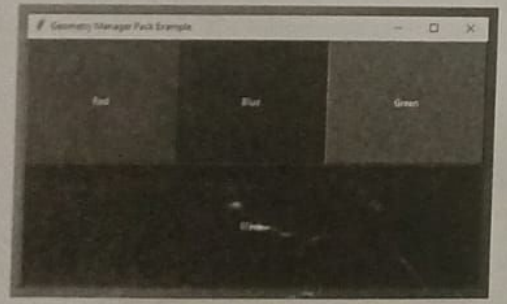
XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

EXAMPLE-2

Placing widgets on top of each other and side by side. We can do this by side option.

```
# Importing tkinter module
from tkinter import *
root = Tk()
root.title("Geometry Manager Pack Example")
topframe = Frame(root)
topframe.pack(side = TOP, expand = True, fill = BOTH)
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM,expand = True, fill = BOTH)
# button widgets which can also expand and fill in the parent widget entirely
b1 = Button(topframe, text = "Red", background = "red", fg = "white")
b1.pack(side = LEFT, expand = True, fill = BOTH)
b2 = Button(topframe, text = "Blue", background = "blue", fg = "white")
b2.pack(side = LEFT, expand = True, fill = BOTH)
b3 = Button(topframe, text = "Green", background = "green", fg = "white")
b3.pack(side = LEFT, expand = True, fill = BOTH)
b4 = Button(bottomframe, text="Black",background = "black", fg="white")
b4.pack(side = BOTTOM, expand = True, fill = BOTH)
root.mainloop()
```

Output :



The grid() Method : This geometry manager organizes widgets in a table-like structure in the parent widget. The master widget is split into rows and columns, and each part of the table can hold a widget. It uses column, columnspan, ipadx, ipady, padx, pady, row, rowspan and sticky.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Syntax :

```
widget.grid( grid_options )
```

column : The column to put widget in. The default column is 0, which is the leftmost column.

columnspan : How many columns widget takes up. The default is 1.

ipadx : How many pixels to pad widget horizontally inside the widget's borders.

ipady : How many pixels to pad widget vertically inside the widget's borders.

padx : How many pixels to pad widget horizontally outside the widget's borders.

pady : How many pixels to pad widget vertically outside the widget's borders.

row : The row to put widget in. The default row is 0, which is the topmost column.

rowspan : How many rows the widget takes up. The default is 1.

sticky : When the widget is smaller than the cell, sticky is used to indicate which sides and corners of the cell the widget sticks to. The direction is defined by compass directions: N, E, S, W, NE, NW, SE and SW and zero.

5.3 USE VARIOUS WIDGETS

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called ***widgets.*** There are currently 15 types of widgets in Tkinter.

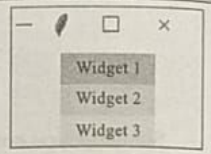
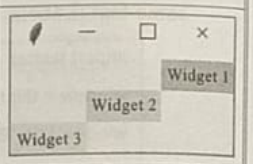
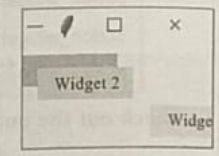
We present these widgets as well as a brief description in the following table.

S.No.	Widget	Description	Example
1.	Button	The Button is used to add various kinds of buttons to the python application. It contains a clickable area with text that calls an associated function whenever the user clicks in the area.	Button
2.	Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.	
3.	Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.	Checkbutton

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following three geometry managers are: pack, grid, and place.

Here is a table showing examples of the different geometry managers :

S.No.	Manager	Description	Example
1.	Pack	Pack is often the easiest geometry manager to use, as it just puts widgets in a single row or column (default). It "packs" the widgets by putting them side-by-side (or top-to-bottom).	
2.	Grid	The grid manager places widgets in a table format with rows and columns. It will avoid overlapping widgets and will resize rows/columns as necessary to fit the widgets.	
3.	Place	The place geometry manager offers the most control but can be the most difficult to use. It allows you to specify the (or) relative positions of the widgets in a window (or parent widget).	

The pack() Method - This geometry manager organizes widgets in blocks before placing them in the parent widget.

Syntax :

```
widget.pack( pack_options )
```

Here is the list of possible options :

expand : When set to true, widget expands to fill any space not otherwise used in widget's parent.

fill : Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), (or) BOTH (fill both horizontally and vertically).

side : Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, (or) RIGHT.

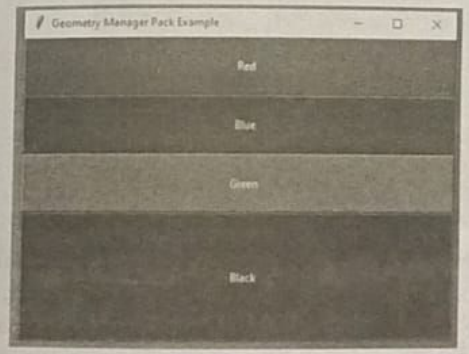
WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

EXAMPLE-1

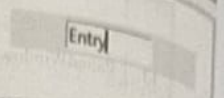
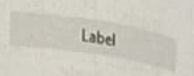
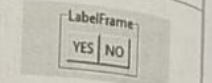
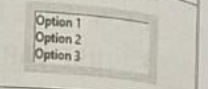
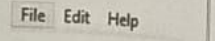
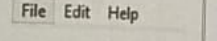
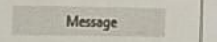
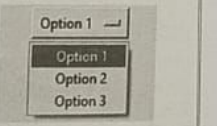
Putting a widget inside frame and filling entire frame. We can do this with the help of expand and fill options.

```
# importingtkinter module
from tkinter import *
root = Tk()
root.title("Geometry Manager Pack Example")
topframe = Frame(root)
topframe.pack(side = TOP, expand = True, fill = BOTH)
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM,expand = True, fill = BOTH)
# button widgets which can also expand and fill in the parent widget entirely
b1 = Button(topframe, text = "Red", background = "red", fg = "white")
b1.pack(side = TOP, expand = True, fill = BOTH)
b2 = Button(topframe, text = "Blue", background = "blue", fg = "white")
b2.pack(side = TOP, expand = True, fill = BOTH)
b3 = Button(topframe, text = "Green", background = "green", fg = "white")
b3.pack(side = TOP, expand = True, fill = BOTH)
b4 = Button(bottomframe, text="Black",background = "black", fg="white")
b4.pack(side = BOTTOM, expand = True, fill = BOTH)
root.mainloop()
```

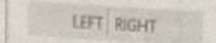
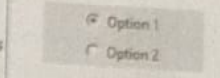
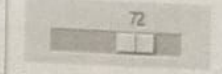
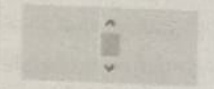
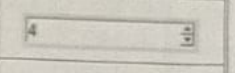
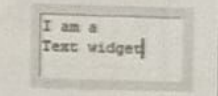
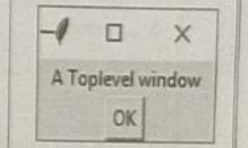
Output :



XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

4.	Entry	The entry widget is used to display the single line line text field for accepting values from a user. For entering multiple lines, use the *Text* widget.	
5.	Frame	It can be defined as a container to which, another widget can be added and organized. A Frame can be useful for grouping other widgets together in a complex layout.	
6.	Label	The Label widget is used to provide a single-line caption for other widgets. It cal also contain images. It is used to display text (or) an image that may not be edited by the user.	
7.	LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer (or) container for complex window layouts.	
8.	Listbox	The ListBox widget is used to display a list of options to the user. The user can choose (highlight) one or more options.	
9.	Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton. Used to create menus and submenus within an interface. Can be used to create the always-shown "menu bar" popular in many GUI applications.	
10.	Menubutton	The Menubutton is used to display the menu items to the user. Obsolete as of Tk 8.0. Use Menu widget instead.	
11.	Message	Used to display static text, like *Label*, but allows for multiple lines, text wrapping and maintaining aspect ratios.	
12.	OptionMenu	Drop-down (or pop-up) menu that allows users to select one option from several options.	

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

13.	PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically. These panes can be resized by the user by dragging the separator line(s) (known as *"sashes"*).	
14.	Radiobutton	The Radiobutton is different from a checkbutton. Here, the user is provided with various options and the user can select only one option among them. Several Radiobuttons can be used together to allow the user to select one option out of a group of options.	
15.	Scale	It is used to provide the slider to the user. User can select a numerical value by moving a slider.	
16.	Scrollbar	It provides the scrollbar to the user so that the user can scroll the window up and down. Paired with a *Canvas*, *Entry*, *Listbox*, (or) *Text* widget to allow for scrolling within that that widget.	
17.	Spinbox	Allows the user to select only one option out of a list of options.	
18.	Text	It is different from Entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it. It can be used as a full text editor by the user.	
19.	Toplevel	A container for other widgets (much like the the *Frame* widget) that appears in its own window. It can be useful for creating other application windows or pop-up notifications. It is used to create a separate window container.	
20.	tkMessageBox	This module is used to display the messagebox in the desktop based applications.	

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

5.4 VARIOUS ATTRIBUTES OF WIDGETS

Button Widget : The button widget is used to add various types of buttons to the python application. Python allows us to configure the look of the button according to our requirements. Various options can be set or reset depending upon the requirements.

We can also associate a method (or) function with a button which is called ***when the button is pressed.***

Syntax :

Here is the simple syntax to create this widget :

w = Button (master, option=value, ...)

Parameters :

master : This represents the parent window.

options : Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

A list of possible options is given below :

S.No.	Option	Description
1.	activebackground	It represents the background of the button when the mouse hover the button.
2.	activeforeground	It represents the font color of the button when the mouse hover the button.
3.	Bd	It represents the border width in pixels.
4.	Bg	It represents the background color of the button.
5.	Command	It is set to the function call which is scheduled when the function is called.
6.	Fg	Foreground color of the button.
7.	Font	The font of the button text.
8.	Height	The height of the button. The height is represented in the number of text lines for the textual lines or the number of pixels for the images.
9.	Highlightcolor	The color of the highlight when the button has the focus.
10.	Image	It is set to the image displayed on the button.
11.	justify	It illustrates the way by which the multiple text lines are represented. It is set to LEFT for left justification, RIGHT for the right justification and CENTER for the center.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

12.	Padx	Additional padding to the button in the horizontal direction.
13.	pady	Additional padding to the button in the vertical direction.
14.	Relief	It represents the type of the border. It can be SUNKEN, RAISED, GROOVE and RIDGE.
15.	State	This option is set to DISABLED to make the button unresponsive. The ACTIVE represents the active state of the button.
16.	Underline	Set this option to make the button text underlined.
17.	Width	The width of the button. It exists as a number of letters for textual buttons or pixels for image buttons.
18.	Wraplength	If the value is set to a positive number, the text lines will be wrapped to fit within this length.

5.5 HANDLE EVENTS GENERATED BY VARIOUS WIDGETS

In programming, an event is something that occurs within an application's environment such as a mouse click, key press (or) changes in the GUI. In Python, using the Tkinter GUI package, we can handle these events by binding them to either predefine (or) user-defined functions, this allows us to run pieces of code when a certain event occurs within a widget.

Basic Tkinter Event Binding Syntax : Tkinter application spends most of its time inside an event loop (entered via the mainloop method). Events can come from various sources, including key presses and mouse operations by the user and redraw events from the window manager.

Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can bind Python functions and methods to events.

```
widget.bind(event, handler)
```

To bind an event with a function, we can make use of the bind() function that is included in all widgets. The bind() function takes 2 arguments :

-event : A representative string that contains details about which event to listen for, this must be given in the following format :

"<modifier-type-detail>". The only required part of this string is the "type" section, this represents the type of event to listen for, if you leave out the other section it can be written as "<type>".

-handler : The name of the function to call when the event occurs. This is only the name of the function, meaning you cannot include a list of your own arguments.

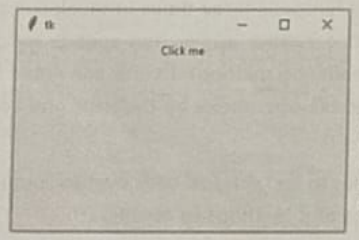
XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

The function included as the event handler will be passed an event object which include details about the event that was triggered, meaning you should include a parameter to be assigned to this object in your function.

As an example, let's assign an event handler to a mouse click event on a label that will print a message when activated :

```
from tkinter import *
window = Tk()
def mouseClick(event):
print( "mouse clicked" )
label = Label( window, text="Click me" )
label.pack()
label.bind( "<Button>", mouseClick )
window.mainloop()
```

The above code will create a window with a single label with the text "click me", when any of the mouse buttons are clicked (left click, right click or middle mouse button click) the mouseClick() function will be called and will print ***"mouse clicked"*** to the console.




```
mouse clicked
mouse clicked
mouse clicked
mouse clicked
mouse clicked
```

Event types : There are several event types available with Tkinter, including :

KeyPress : Activated when a keyboard button has been pressed, the **Key** event can also be used for this.

KeyRelease : Activated when a keyboard button is released.

Button : Activated when a mouse button has been clicked.

ButtonRelease : Activated when a mouse button has been released.

Motion : Activated when the mouse cursor moves across the designated widget.
Enter : Activated when the mouse cursor enters the designated widget.

Leave : Activated when the mouse cursor leaves the designated widget.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

MouseWheel : Activated when the mouse wheel is scrolled.

FocusIn : Activated when the designated widget gains focus through user input such as the mouse clicking on it.

FocusOut : Activated when the designated widget loses focus.

Configure : Activated when the designated widget's configurations have changes such as its width being adjusted by the user (or) its border being adjusted.

Event modifiers : An event modifier can alter the circumstances in which an event's handler is activated, for example, some modifiers will require another button to be depressed while the event occurs.

Control : Requires that the control button is being pressed while the event is occurring.

Alt : Requires that the alt button is being pressed while the event is occurring.

Shift : Requires that the shift button is being pressed while the event is occurring.

Lock : Requires that caps lock is activated when the event occurs.

Double : Requires that the given event happens twice in quick succession (such as a doubleclick).

Triple : Requires that the given event happens three times in quick succession.

Quadruple : Requires that the given event happens four times in quick succession.

As an example, let's create an event handler that only activates on a double click :

```
label.bind( "<Double-Button>", mouseClick )
```

Event details : The details section of the event string allows us to specify a more specific event such as only a certain key on the keyboard being pressed or only a certain mouse being pressed.

-When using Button (or) ButtonRelease we can give a numeric detail from 1 to 5 which represent the specific mouse button you wish to have the handler trigger from.

-When using Key, KeyPress (or) KeyRelease we can give the ASCII value of the specific key we wish to trigger the event.

As an example, let's create an event handler that only activates on the double click of the left mouse button.

```
label.bind( "<Double-Button-1>", mouseClick )
```

The event object The event object that is passed to the handler when the event is triggered can be used to collect and use information about the event that has occurred. The event object has a number of useful properties such as :

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

keysym : Returns the name of the key (space, e, return) that triggered a keyboard based event such as KeyPress, Key (or) KeyRelease.

-keycode : Returns the code of the key that triggered a keyboard based event.

-button : Returns the mouse button (1-5) that triggered a mouse based event.

-x : Returns the x coordinate of where events such as Button occur.

-y : Returns the y coordinate of where events such as Button occur.

-width : Returns the current width of the widget associated with the event.

-height : Returns the current height of the widget associated with the event.

As an example, let's create an event handler that prints the x and y coordinates of a mouse click event to the console.

```
defmouseClick( event ):
    print("mouse clicked at x=" + event.x + " y=" + event.y)
    label.bind( "<Button>", mouseClick )
Potential output of the code could be:
"mouse clicked at x=45 y=23"
```

If an event matching the event description occurs in the widget, the given *handler* is called with an ***object describing the event.***

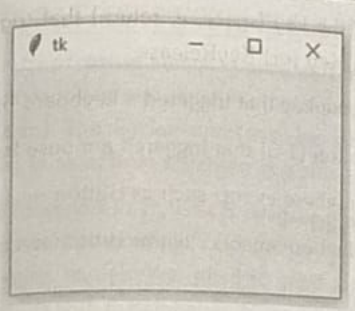
Heres a simple example :

Capturing clicks in a window :

```
from tkinter import *
root = Tk()
def callback(event):
print("clicked at", event.x, event.y)
frame = Frame(root, width=100, height=100)
frame.bind("<Button-1>", callback)frame.pack()
root.mainloop()
```

In this example, we use the bind method of the frame widget to bind a callback function to an event called **<Button-1>.** Run this program and click in the window that appears. Each time you click a message like "clicked at 44 63" is printed to the console window.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL



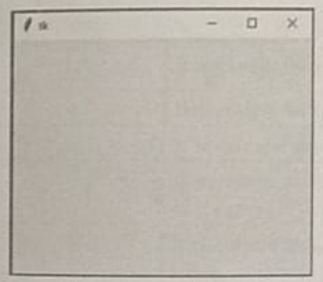

```
clicked at 42 30
clicked at 21 27
clicked at 32 78
clicked at 85 82
clicked at 83 28
clicked at 25 6
clicked at 32 60
clicked at 98 55
clicked at 26 51
clicked at 26 51
clicked at 50 72
clicked at 49 72
clicked at 14 94
```

Keyboard events are sent to the widget that currently owns the keyboard focus. You can use the focus_set method to move focus to a widget :

Capturing Keyboard Events :

```
from tkinter import *
root = Tk()
def key(event):
print("pressed", repr(event.char))
def callback(event):
frame.focus_set()
print("clicked at", event.x, event.y)
frame = Frame(root, width=100, height=100)
frame.bind("<Key>", key)
frame.bind("<Button-1>", callback)
frame.pack()
root.mainloop()
```

If you run this script, you ll find that you have to click in the frame before it starts receiving any keyboard events.




```
clicked at 34 21
clicked at 70 74
pressed 'w'
pressed 'e'
pressed 'l'
pressed 'c'
pressed 'o'
pressed 'm'
pressed 'e'
```

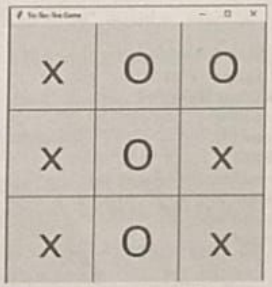
XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

EXAMPLE-1

Write a GUI program for Tic-Tac-Toe Game to handle events.

```
from tkinter import *
def callback(r,c):
    global player
    if player=='X' and states[r][c]==0:
        b[r][c].configure(text='x')
        states[r][c]='X'
        player='O'
    if player=='O' and states[r][c]==0:
        b[r][c].configure(text='O')
        states[r][c]='O'
        player='X'
root=Tk()
root.title("Tic-Tac-Toe Game")
states=[[0,0,0],[0,0,0],[0,0,0]]
b=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(3):
    for j in range(3):
        b[i][j]=Button(font=('verdana',56),width=3,bg='yellow',command=lambda
r=i,c=j:callback(r,c))
        b[i][j].grid(row=i,column=j)
player='X'
root.mainloop()
```

Output :



5.6 CREATE PATTERNS USING REGULAR EXPRESSION

The Regular Expression, also known as ***RegEx*** in python is a sequence of metacharacters, numbers and alphabets that define a particular pattern (email, password, username pattern). The regular expressions can be defined as the sequence of characters which are used to search for a pattern in a string.

The regular expression in Python is used to identify whether a pattern exists in the given string or not. For example, consider yourself as a Manager and you have a list of all your employee names. You need to send a common message to the employees whose name starts with the alphabet 'h', ends with the alphabet 'i' and of length 6. The below-given RegEx helps you to do this.

```
^h....i$
```

The pattern is, any six-letter string starting with the alphabet 'h' and ending with 'i'. Here, the symbols '^' and '$' are called ***metacharacters.*** Below is the matching status of various names.

S.No.	Pattern	Names	Status
1.	^h.....i$	harini	Matched
2.	^h.....i$	harsit	Not Matched
3.	^h....i$	hrtyei	Matched

The module ***re*** provides the support to use regex in the python program. The re module throws an exception if there is some error while using the regular expression.

The re module must be imported to use the regex functionalities in python.

```
import re
```

Forming a Regular Expression : A regular expression can be formed by using the mix of meta-characters, special sequences and sets.

Meta-Characters : Metacharacter is a character with the specified meaning.

S.No.	Metacharacter	Description	Example
1.	[]	It represents the set of characters.	"[a-z]"
2.	\	It represents the special sequence.	"\r"
3.	.	It signals that any character is present at some	"hel.o."
4.	^	It represents the pattern present at the beginning of the string.	"^hello"
5.	$	It represents the pattern present at the end of the string.	"welcome"

EXAMPLE

Suppose you wish to look for the occurrence of a particular sub-string in a string :

```
import re
str = "at what time?"
match = re.search('at',str)
if (match):
print("String found at: " ,match.start())
else:
print("String not found!")
```

Output :

```
String found at:  0
```

In above example the start() function returns the start index of the matched string.

```
import re
str = "at what time?"
match = re.search('of',str)
if (match):
print("String found at: " ,match.start())
else:
print("String not found!")
```

Output :

```
String not found!
```

3. **re.findall() :** The re.findall() function returns a list of strings containing all matches of the specified pattern. It returns the patterns in the order they are found. If there are no matches, then an empty list is returned.

The function takes as input the following :

- A character pattern.
- The string from which to search.

Consider the following example.

```
import re
str = "How are you. How is everything"
matches = re.findall("How", str)
print(matches)
```

Output :

```
['How', 'How']
```

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

4. **re.split() :** The re.split() function splits the string at every occurrence of the sub-string and returns a list of strings which have been split.

EXAMPLE

Suppose we wish to split a string wherever there is an occurrence of a,

```
import re
str = "at what time?"
match = re.split('a',str)
print(match)
```

Output :

```
['', 't wh', 't time?']
```

In case there is no match, the string will be returned as it is, in a list.

```
import re
str = "at what time?"
match = re.split('z', str)
print(match)
```

Output :

```
['at what time?']
```

5. **re.sub() :** The re.sub() function is used to replace occurrences of a particular sub-string with another sub-string.

This function takes as input the following :

- The sub-string to replace.
- The sub-string to replace with,
- The actual string.

EXAMPLE

Suppose you wish to insert !!! instead of a white-space character in a string. This can be done via the re.sub() function as follows :

```
import re
str = "at what time?"
match = re.sub("\s","!!!",str)
print(match)
```

Output :

```
at!!!what!!!time?
```

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

6.	*	It represents zero (or) more occurrences of a pattern in the string.	"hello*"
7.	+	It represents one or more occurrences of a pattern in string.	"hello+"
8.	{}	The specified number of occurrences of a pattern the string.	"hello{2}"
9.	|	It represents either this or that character is present.	"hello|world"
10.	()	Capture and group.	

Special Sequences : Special sequences are the sequences containing\followed by one of the characters.

S.No.	Character	Description
1.	\A	It returns a match if the specified characters are present at the beginning of the string.
2.	\b	It returns a match if the specified characters are present at the beginning (or) the end of the string.
3.	\B	It returns a match if the specified characters are present at the beginning of the string but not at the end.
4.	\d	It returns a match if the string contains digits [0-9].
5.	\D	It returns a match if the string doesn't contain the digits [0-9].
6.	\s	It returns a match if the string contains any white space character.
7.	\S	It returns a match if the string doesn't contain any white space character.
8.	\w	It returns a match if the string contains any word characters.
9.	\W	It returns a match if the string doesn't contain any word.
10.	\Z	Returns a match if the specified characters are at the end of the string.

Sets : A set is a group of characters given inside a pair of square brackets. It represents the special meaning.

S.No.	Set	Description
1.	[arn]	Returns a match if the string contains any of the specified characters in the set.
2.	[a-n]	Returns a match if the string contains any of the characters between a to n.
3.	[^arn]	Returns a match if the string contains the characters except a, r, and n.
4.	[0123]	Returns a match if the string contains any of the specified digits.
5.	[0-9]	Returns a match if the string contains any digit between 0 and 9.
6.	[0-5][0-9]	Returns a match if the string contains any digit between 00 and 59.
7.	[a-zA-Z]	Returns a match if the string contains any alphabet (lower-case or upper case).

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Regex Functions : The re module provides users a variety of functions to search for a pattern in a particular string. Below are some of the most frequently used functions in detail :

S.No.	Function	Description
1.	match	This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false.
2.	search	This method returns the match object if there is a match found in the string.
3.	findall	It returns a list that contains all the matches of a pattern in the string.
4.	split	Returns a list in which the string has been split in each match.
5.	sub	Replace one (or) many matches in the string.

1. **re.match() :** To check whether the given string matches with the given RegEx pattern, we need to use a method named 'match()'. This method is pre-defined in the module named 're'. In the below example, the string 'harini' is compared with the RegEx pattern '^h....i$'.

EXAMPLE

```
import re
pattern = '^h....i$' #RegEx pattern
str1 = 'harini' #string to be matched with RegEx pattern
res = re.match(pattern, str1) #result of the match
if(res):
print("Pattern Matched")
else:
print("Pattern Not Matched")
```

Output :

```
Pattern Matched
```

2. **re.search() :** The re.search() function returns a match object in case a match is found. In case of more than one match, the first occurrence of the match is returned. If no occurrence is found, None is returned.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

5.7 VALIDATE DATA USING REGULAR EXPRESSION

Validate an Interger : Regexp work on the character base and \d means a single digit 0...9 and not a decimal number.

A regular expression that matches only integers with a sign could be for example.

```
^[-+]?[0-9]+$
```

Meaning :

- ^ - start of string.
- [-+]? - an optional (this is what ? means) minus (or) plus sign.
- [0-9]+ - one or more digits (the plus means "one or more" and [0-9] is another way to say\d).
- $ - end of string.

```
import re
number =input("Please enter a number: ")
if not re.match("^[-+]?[0-9]+$", number):
    print ("Error! Make sure you only use numbers")
    number = input("Please enter a number: ")
print("You picked number "+ number)
```

Output :

```
Please enter a number: -123456ty
Error! Make sure you only use numbers
Please enter a number: -12345
You picked number -12345
```

Validates a floating point number : #Python program to check input is Floating point number (or) not

```
import re
regex = '[+-]?[0-9]+\.[0-9]+'
def check(floatnum):
    if(re.search(regex, floatnum)):
            print("Your Floating point number is",floatnum)
    else:
            print("Error! Make sure you only use Floating point number")
```



```
if __name__ == '__main__':
    floatnum =input("Please enter a float number: ")
    check(floatnum)
```

Output :

```
Please enter a float number: 12.345
Your Floating point number is 12.345
Please enter a float number: -34.567
Your Floating point number is -34.567
Please enter a float number: 456
Error! Make sure you only use Floating point number
```

Validate a string :

```
import re
str =input("Please enter a string: ")
if not re.match("^[a-zA-Z0-9]+$", str):
    print ("Error! Make sure you only use alphanumeric characters")
    str =input("Please enter a string: ")
print("Your string is:",str)
```

Output :

```
Please enter a string: welcome@123^%$
Error! Make sure you only use alphanumeric characters
Please enter a string: welcome567
Your string is: welcome567
```

Validate a Password : Lets take a password as a combination of alphanumeric characters along with special characters, and check whether the password is valid or not with the help of few conditions.

Conditions for a valid password are :

- Should have at least one number 0-9 (?=.*\d).
- Should have at least one uppercase and one lowercase character (?=.*[a-z]) (?=.*[A-Z]).
- Should have at least one special symbol (?=.*[@#$%^&+=]).
- At least 8 characters (?=.{8,}).

```
import re
pattern = "^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$"
password = input("Enter password to test: ")
result = re.findall(pattern, password)
if (result):
print("Valid password")
else:
print("Password not valid")
```

Output :

```
Enter password: welcome@3
Invalid password
Enter password: WelCome#67
Password is valid
```

Validate Phone Number : The number which satisfies the below criteria, is a valid mobile Number.

- The first digit should contain a number between 7 to 9.
- The rest 9 digits can contain any number between 0 to 9.
- The mobile number can have 11 digits also by including 0 at the starting.
- The mobile number can be of 12 digits also by including 91 at the starting.

```
import re
pattern = "(0/91)?[7-9][0-9]{9}"
mobile = input("Enter phone number: ")
result = re.findall(pattern, mobile)
if (result):
print("Valid phone number is: ", mobile)
else:
print("please enter a valid mobile number")
```

Output :

```
Enter phone number: 0768956
please enter a valid mobile number
Enter phone number: 09985678789
Valid phone number is:  09985678789
Enter phone number: 919440678890
Valid phone number is:  919440678890
```

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Validate an email Address : There are times when you want to validate email address. You can write your own code to validate email address but there are many regular expressions which you can use to validate email address.

```
import re
def isValid(email):
  if(re.match("^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$", email) != None):
    return True
  return False
email=input("Enter email address:")
if(isValid(email) == True):
print("This is a valid email address")
else:
print("This is not a valid email address")
```

Output :

```
Enter email address:suresh@gmail
This is not a valid email address
Enter email address:suresh_123@yahoo
This is not a valid email address
Enter email address:suresh_babu.123@gmail.com
This is a valid email address
```

Validate Dates : #Date Format (dd/mm/yyyy) Regular Expression Pattern

```
import re
def isValid(date):
  if(re.match("(0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\\d\\d)", date) != None):
    return True
  return False
date=input("Enter date:")
if(isValid(date) == True):
print("This is a valid Date")
else:
print("This is not a valid Date")
```

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

Output :

```
Enter date:34/45/2020
This is not a valid Date
Enter date:31/03/2020
This is a valid Date
Enter date:32/10/2019
This is not a valid Date
```

Validate an IP Address : Every computer connected to the Internet is identified by a unique four-part string, known as ***its Internet Protocol (IP) address***. An IP address (version 4) consists of four numbers (each between 0 and 255) separated by periods. The format of an IP address is a 32-bit numeric address written as four decimal numbers (called ***octets***) separated by periods; each number can be written as 0 to 255. (**E.g.** 0.0.0.0 to 255.255.255.255).

```
# Python program to validate an Ip address
import re
regex = ""^(25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)\.(
        25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)\.(
        25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)\.(
        25[0-5]|2[0-4][0-9]|[0-1]?[0-9][0-9]?)""
def check(Ip):
    if(re.search(regex, Ip)):
            print("Valid Ip address")
    else:
            print("Invalid Ip address")
if __name__ == '__main__' :
    # Enter the Ip address
    Ip = "192.168.0.1"
    check(Ip)
    Ip = "110.234.52.124"
    check(Ip)
    Ip = "366.1.2.2"
    check(Ip)
```

Output :

```
Valid Ip address
Valid Ip address
Invalid Ip address
```

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

OBJECTIVE TYPE QUESTIONS

1. **In python context, what does GUI stand for?** [ ]
   (a) General User Interface (b) Graphical Unit Interface
   (c) Golfing Union of Ireland (d) Graphical User Interface

2. **Which of the following are valid Tkinter widgets?** [ ]
   (a) Entry (b) Button
   (c) Label (d) ColorPicker

3. **Which of the following geometry managers are available in Tkinter?** [ ]
   (a) .grid() (b) .place()
   (c) .pack() (d) .button()

4. **For what purpose, the bg is used in the Tkinter widget?** [ ]
   (a) To change the size of the widget
   (b) To change the color of the widget
   (c) To change the direction of the widget
   (d) To change the background of the widget

5. **What is Tk() in Tkinter python?** [ ]
   (a) It is a widget (b) It is a function
   (c) It is a constructor (d) All of the above

6. **For user entry data, which widget we use in Tkinter?** [ ]
   (a) Entry (b) Text
   (c) Both of the above (d) None of the above

7. **Which module in Python supports regular expressions?** [ ]
   (a) Re (b) regex
   (c) pyregex (d) None of the mentioned

8. **The function re.error raises an exception if a particular string contains no match for the given pattern.** [ ]

```
def foo():
```

(a) True
(b) False

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

CHAPTER 6

DATA PROCESSING AND PROGRAMMING RASPBERRY PI

Chapter Outline

6.1 HANDLE OPEN, CLOSE, READ, WRITE AND APPEND OPERATIONS ON FILES USING PROGRAMS

If you are working in a large software application where they process a large number of data, then we cannot expect those data to be stored in a variable as the variables are volatile in nature. Hence when are you about to handle such situations, the role of files will come into the picture. As files are non-volatile in nature, the data will be stored permanently in a secondary device like Hard Disk and using python we will handle these files in our applications.

Let's take an Example of how normal people will handle the files. If we want to read the data from a file or write the data into a file, then, first of all, we will open the file or will create a new file if the file does not exist and then perform the normal read/write operations, save the file and close it. Similarly, we do the same operations in python using some in-built methods or functions.

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files.

Binary files in Python

Most of the files that we see in our computer system are called ***binary files.***

Example binary files :

- **Document files:** .pdf, .doc, .xls etc.,
- **Image files:** .png, .jpg, .gif, .bmp etc.,
- **Video files:** .mp4, .3gp, .mkv, .avi etc.,
- **Audio files:** .mp3, .wav, .mka, .aac etc.,
- **Database files:** .mdb, .accde, .frm, .sqlite etc.,
- **Archive files:** .zip, .rar, .iso, .7z etc..
- **Executable files:** .exe, .dll, .class etc.,

All binary files follow a specific format. We can open some binary files in the normal text editor but we can't read the content present inside the file. That's because all the binary files will be encoded in the binary format, which can be understood only by a computer (or) machine. For handling such binary files we need a specific type of software to open it.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

For Example, You need Microsoft word software to open .doc binary files. Likewise, you need a pdf reader software to open .pdf binary files and you need photo editor software to read the image files and so on.

Text files in Python : Text files don't have any specific encoding and it can be opened in normal text editor itself.

Example Text files :

- **Web standards :** html, XML, CSS, JSON etc.
- **Source code :** c, app, js, py, java etc.
- **Documents :** txt, tex, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

Python File Handling Operations : Most importantly there are 4 types of operations that can be handled by Python on files :

- Open a file
- Read a file
- Write into file
- Close a file

Other Operations Include :

- Rename a file.
- Delete a file.

Python Create and Open a File : Python has an in-built function called **open()** to open a file. No module is required to be imported for this function.

It takes a minimum of one argument as mentioned in the below syntax. The open method returns a file object which is used to access the write, read and other in-built methods.

Syntax :

```
file_object = open(file_name, mode)
```

Here, file_name is the name of the file or the location of the file that you want to open, and file_name should have the file extension included as well. Which means in **test.txt** – the term test is the name of the file and .txt is the extension of the file.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

```
file1.seek(0)
# readlines function
print("Output of Readlines function is ")
print(file1.readlines())
file1.close()
```

Output :

```
Output of Read function is
Welcome
Hello World
Hello Python
Good morning
How are you
Output of Readline function is
Welcome
Output of Read(9) function is
Welcome
Output of Readline(9) function is
Hello Wor
Output of Readlines function is
['Welcome \n', 'Hello World\n', 'Hello Python\n', 'Good morning \n', 'How are you \n']
```

Python Write to File : In order to write data into a file, we must open the file in write mode. We need to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.

We have two methods for writing data into a file as shown below.

write(string) : Inserts the string str1 in a single line in the text file.

```
File_object.write(str1)
```

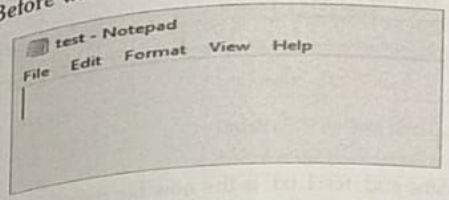
writelines(list) : For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

```
File_object.writelines(L) for L = [str1, str2, str3]
```

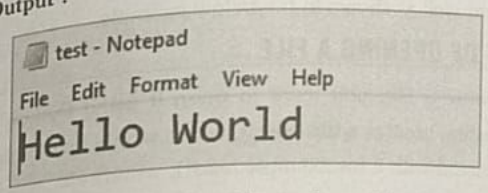
WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Note : If you are opening a file test.txt, it will automatically create a file named test.txt in the current directory where you are creating the python file for example *C:\Users\user\AppData\Local\Programs\Python\Python38* directory, otherwise if you want to create a file in a specific directory you want you have to mention the full path for example *C:/Documents/Python/test.txt*

Before writing data to a test.txt file :



Output :



Python Close File : In order to close a file, we must first open the file. In python, we have an in-built method called **close()** to close the file which is opened.

Whenever you open a file, it is important to close it, especially, with write method. Because if we don't call the close function after the write method then whatever data we have written to a file will not be saved into the file.

EXAMPLE-1

```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.read())
my_file.close()
```

EXAMPLE-2

```
my_file = open("C:/Documents/Python/test.txt", "w")
my_file.write("Hello World")
my_file.close()
```

Python Rename (or) Delete File : Python provides us with an "os" module which has some in-built methods that would help us in performing the file operations such as renaming and deleting the file.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

The mode in the open function syntax will tell Python as what operation you want to do on a file.

- **'r' – Read Mode:** Read mode is used only to read data from the file.
- **'w' – Write Mode:** This mode is used when you want to write data into the file (or) modify it. Remember write mode overwrites the data present in the file.
- **'a' – Append Mode:** Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
- **'r+' – Read or Write Mode:** This mode is used when we want to write or read the data from the same file.
- **'a+' – Append or Read Mode:** This mode is used when we want to read data from the file (or) append the data into the same file.

Note : The above-mentioned modes are for opening, reading or writing text files only.

While using binary files, we have to use the same modes with the letter **'b'** at the end. So that Python can understand that we are interacting with binary files.

- **'wb'** – Open a file for write only mode in the binary format.
- **'rb'** – Open a file for the read-only mode in the binary format.
- **'ab'** – Open a file for appending only mode in the binary format.
- **'rb+'** – Open a file for read and write only mode in the binary format.
- **'ab+'** – Open a file for appending and read-only mode in the binary format.

EXAMPLE-1

```
fo = open("C:/Documents/Python/test.txt", "r+")
```

In the above example, we are opening the file named 'test.txt' present at the location 'C:/Documents/Python/' and we are opening the same file in a read-write mode which gives us more flexibility.

EXAMPLE-2

```
fo = open("C:/Documents/Python/img.bmp", "rb+")
```

In the above example, we are opening the file named 'img.bmp' present at the location "C:/Documents/Python/", But, here we are trying to open the binary file.

Python Read From File

In order to read a file in python, we must open the file in read mode.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

There are Three Ways to Read Data from a Text File :

read() : Returns the read bytes in form of a string. Reads n bytes, if n is not specified, reads the entire file.

```
File_object.read([n])
```

readline() : Reads a line of the file and returns in the form of a string. For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

```
File_object.readline([n])
```

readlines() : Reads all the lines and return them as each line a string element in a list.

```
File_object.readlines()
```

Note : '\n' is treated as a special character of two bytes

```
# Program to show various ways to read and write data in a file.
file1 = open("myfile.txt","w")
L = ["Hello World\n","Hello Python\n","Good morning \n", "How are you \n"]#\n is placed to indicate EOL (End of Line)
file1.write("Welcome \n")
file1.writelines(L)
file1.close()
file1 = open("myfile.txt","r+")#to change file access modes
print("Output of Read function is ")
print(file1.read())
# seek(n) takes the file handle to the nth bite from the beginning.
file1.seek(0)
print("Output of Readline function is ")
print(file1.readline())
file1.seek(0)
# To show difference between read and readline
print("Output of Read(9) function is ")
print(file1.read(9))
print("Output of Readline(9) function is ")
print(file1.readline(9))
print()
```

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

In order to use this module, first of all, we need to import the "os" module in our program and then call the related methods.

rename() method : This rename() method accepts two arguments i.e., the current file name and the new file name.

Syntax :

```
os.rename(current_file_name, new_file_name)
```

EXAMPLE-1

```
import os
os.rename("test.txt", "test1.txt")
```

Here 'test.txt' is the current file name and 'test1.txt' is the new file name.

You can specify the location as well as shown in the below example.

6.2 DIFFERENT MODES OF OPENING A FILE

Before you can read (or) write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it for example read(), write() etc.,

Syntax :

```
file_object = open(file_name [, access_mode][, buffering])
```

Here are parameter details :

- **file_name** : The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** : The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc., A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering** : If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default (default behavior).

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Here is a list of the different modes of opening a file :

S.No.	Access Modes	Description
1.	r	Opens a file for reading only. The file pointer is placed at the beginning of the default mode.
2.	rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3.	r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4.	rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5.	w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6.	wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7.	w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8.	wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9.	a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10.	ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11.	a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12.	ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

6.3 DELETE FILES AND FOLDERS

Here we are going to discuss how to delete a file or folder in Python. The process of removing a file (or) folder in Python is straightforward using the **os module**.

- os.remove Deletes a single file.
- os.rmdir Deletes an empty folder.
- shutil.rmtree Deletes a folder and all its contents.

1. **Deletes a Single File :** To delete a single file from a directory we are using the os.remove() method.

```
import os
# getting the filename from the user
file_path = input("Enter filename:- ")
# checking whether file exists or not
if os.path.exists(file_path):
os.remove(file_path)
print("File deleted successfully")
else:
print("File not found in the directory")
```

Output :

```
Enter filename:- C:\Users\user\Desktop\test.txt
File deleted successfully
Enter filename:- sample.txt
File not found in the directory
```

2. **Deletes an Empty Folder :** The folder which we are going to delete must be empty. Python will show a warning stating that the folder is not empty. Before removing a folder, make sure that it is empty. We can get the list of files present in the directory using os.listdir() Method. From that, we can check whether the folder is empty (or) not.

```
import os
folder_path = input("Enter folder path:- ")
if os.path.exists(folder_path):
    # checking whether the folder is empty (or) not
    if len(os.listdir(folder_path)) == 0:
os.rmdir(folder_path)
print("Empty folder deleted successfully")
else:
print("Folder is not empty")
else:
print("Folder not found in the folder path")
```

Output :

```
Enter folder path:- C:\Users\user\Desktop\sample
Folder is not empty
Enter folder path:- C:\Users\user\Desktop\sample1
Empty folder deleted successfully
Enter folder path:- sample
Folder not found in the given folder path
```

3. **Deletes a Folder and all its Contents :** Now as shown in below figure I stored the 'Cars.docx' and 'D1.jpg' files within the 'sample' folder which is not an empty folder. To delete this folder, and the files within it, we are using shutil.rmtree () method.

```
import os
import sys
import shutil
# Get folder name
mydir= input("Enter folder name: ")
try:
shutil.rmtree(mydir)
print("folder and all its contents deleted")
except OSError as e:
    print("Error: %s - %s." % (e.filename, e.strerror))
```

Output :

```
Enter folder name: C:\Users\user\Desktop\sample1
Error: C:\Users\user\Desktop\sample1 - The system cannot find the path specified.
Enter folder name: C:\Users\user\Desktop\sample
folder and all its contents deleted
```

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

6.4 CONNECT TO MYSQLDATABASE

To build the real world applications, connecting with the databases is the necessity for the programming languages. However, python allows us to connect our application to the databases like MySQL, SQLite, MongoDB, and many others. Here we will discuss Python - MySQL connectivity and we will perform the database operations in python.

Before you can access MySQL databases using Python, you must install one (or more) of the following packages :

MySQL-python : This package contains the MySQLdb module, which is written in C. It is one of the most commonly used Python packages for MySQL.

To install the MySQL-python package, type the following command :

```
pip install MySQL-python
```

mysql-connector-python : This package contains the mysql.connector module, which is written entirely in Python.

To install the mysql-connector-python package, type the following command:

```
pip install mysql-connector-python
```

PyMySQL : This package contains the pymysql module, which is written entirely in Python. It is designed to be a drop-in replacement for the MySQL-python package.

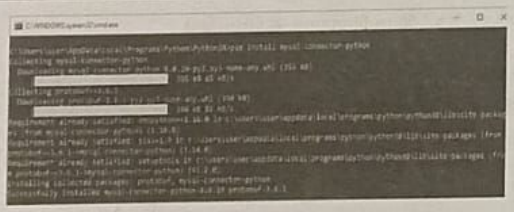
To install the pymysql package, type the following command :

```
pip install pymysql
```

Install mysql.connector : To connect the python application with the MySQL database, we must import the mysql.connector module in the program. The mysql.connector is not a built-in module that comes with the python installation. We need to install it to get it working.

Execute the following command to install it using pip installer.

```
> pip install mysql-connector-python
```

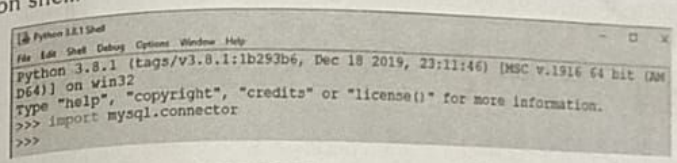


This will take a bit of time to install mysql-connector for python.

Connecting and Creating Databases : Here we will discuss the steps to connect the python application to the database. These are the following steps to connect a python application to our database.

- Import mysql.connector module.
- Create the connection object.
- Create the cursor object.
- Creating New Database.

Import mysql.connector module : We can verify the installation of mysql.connector python module once the process gets over by importing mysql-connector on the python shell.



As we did not get any error messages, we can conclude that we successfully installed mysql-connector for python on our system.

Creating the Connection Object : To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used. Pass the database details like Hostname, username, and the database password in the method call. The method returns the connection object.

The syntax to use the connect() is given below.

```
Connection-Object= mysql.connector.connect(host = <host-name>, user = <username>,
passwd = <password>)
```

Consider the following example.

Example :

```
import mysql.connector as mysql
#Create the connection object
db = mysql.connect(
  host = "localhost",
  user = "root",
  passwd = "dbms"
)
```

```
#printing the connection object
print(db)
```

Output :

```
<mysql.connector.connection_cext.CMySQLConnection object at
0x0000019A5BD3E400>
```

Creating a Cursor Object : The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

The syntax to create the cursor object is given below.

```
<my_cur>  =conn.cursor()
```

EXAMPLE

```
import mysql.connector as mysql
#Create the connection object
db = mysql.connect(
  host = "localhost",
  user = "root",
  passwd = "mysql20"
)
#printing the connection object
print(db)
#creating the cursor object
cursor = db.cursor()
print(cursor)
```

Output :

```
<mysql.connector.connection_cext.CMySQLConnection object at
0x00000210E4E49B20>
CMySQLCursor: (Nothing executed yet)
```

Creating New Database : Getting the list of existing databases

We can get the list of all the databases by using the following MySQL query.

```
> show databases;
```

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

EXAMPLE

```
import mysql.connector as mysql
db = mysql.connect(
  host = "localhost",
  user = "root",
  passwd = "mysql20"
)
cursor = db.cursor()
## executing the statement using 'execute()' method
cursor.execute("SHOW DATABASES")
databases = cursor.fetchall() ## it returns a list of all databases present
print(databases)
for database in databases:
  print(database)
```

Output :

```
[('information_schema',), ('mysql',), ('performance_schema',), ('sakila',),
('studentdb',), ('sys',), ('world',)]
('information_schema',)
('mysql',)
('performance_schema',)
('sakila',)
('studentdb',)
('sys',)
('world',)
```

Creating the New Database : The new database can be created by using the following SQL query.

```
> create database <database-name>
```

EXAMPLE

```
import mysql.connector as mysql
db = mysql.connect(
  host = "localhost",
  user = "root",
  passwd = "mysql20"
```

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

```
)
cursor = db.cursor()
## creating a databse called 'empDB'
cursor.execute("CREATE DATABASE empDB")
mycursor=db.cursor()
mycursor.execute("SHOW DATABASES")
for x in mycursor:
  print(x)
```

Output :

```
('empdb',)
('information_schema',)
('mysql',)
('performance_schema',)
('sakila',)
('studentdb',)
('sys',)
('world',)
```

6.5 PERFORM CREATION OF TABLE, INSERT A ROW IN A TABLE, UPDATE AN ENTRY IN A TABLE AND EXECUTE STORED PROCEDURES

Creating the Table : Here, we will create the new table Employee. We have to mention the database name while establishing the connection object. We can create the new table by using the CREATE TABLE statement of SQL. In our database empDB, the table Employee will have the four columns, i.e., name, id, salary, and department_id initially.

The following query is used to create the new table Employee.

```
> create table Employee (name varchar(20) not null, id int primary key, salary float not
null, Dept_Id int not null)
```

EXAMPLE

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "localhost",user = "root",passwd =
"mysql20",database = "empDB")
#creating the cursor object
cur = myconn.cursor()
```



```
cur.execute("create table Employee(name varchar(20) not null, id int(20) not null primary
key, salary float not null, Dept_id int not null)")
mycursor = myconn.cursor()
mycursor.execute("DESC Employee")
print(mycursor.fetchall())
myconn.close()
```

Output :

```
[('name', 'varchar(20)', 'NO', '', None, ''), ('id', 'int', 'NO', 'PRI', None, ''), ('salary',
'float', 'NO', '', None, ''), ('Dept_id', 'int', 'NO', '', None, '')]
```

```
MySQL  localhost:33060+ ssl  empdb  SQL > use empDB
Default schema set to `empDB`.
Fetching table and column names from `empdb` for auto-completion... Press ^C to stop.
MySQL  localhost:33060+ ssl  empdb  SQL > show tables;
+---------------+
| Tables_in_empdb |
+---------------+
| employee      |
+---------------+
1 row in set (0.0013 sec)
MySQL  localhost:33060+ ssl  empdb  SQL > desc Employee;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | NO   |     | NULL    |       |
| id      | int         | NO   | PRI | NULL    |       |
| salary  | float       | NO   |     | NULL    |       |
| Dept_id | int         | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.0024 sec)
```

Alter Table : Sometimes, we may forget to create some columns, or we may need to update the table schema. The alter statement used to alter the table schema if required. Here, we will add the column branch_name to the table Employee. The following SQL query is used for this purpose.

alter table Employee add branch_namevarchar(20) not null

Consider the following example.

EXAMPLE

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "localhost",user = "root",passwd =
"mysql20",database = "empDB")
#creating the cursor object
cur = myconn.cursor()
cur.execute("alter table Employee add branch_name varchar(20) not null")
myconn.close()
```

Output :

```
MySQL  localhost:33060+ ssl  empdb  SQL > desc Employee;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name        | varchar(20) | NO   |     | NULL    |       |
| id          | int         | NO   | PRI | NULL    |       |
| salary      | float       | NO   |     | NULL    |       |
| Dept_id     | int         | NO   |     | NULL    |       |
| branch_name | varchar(20) | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.0471 sec)
```

6.6 STORE IMAGES INTO DATABASE USING BLOB DATA TYPE

A BLOB (Large Binary OBject) is a MySQL data type that can be used to store binary data. We can convert our files and images into binary data in Python and store them in MySQL table using BLOB.

Note : To insert file or image into MySQL table we need to create a column that has a BLOB as a type. MySQL has the following four BLOB types. Each holds a variable amount of data.

- TINYBLOB
- BLOB
- MEDIUMBLOB
- LONGBLOB

Above BLOB types differ only in the maximum length of the values they can hold.

To Store BLOB data in MySQL table, we need to create a table that can hold binary data. Alternatively, if you have a table then modify it and add one extra column with BLOB as its data type.

You can use the following query to a create table with a BLOB column.

CREATE TABLE Python_Employee(id INT NOT NULL, name VARCHAR(30) NOT NULL, photo BLOB NOT NULL, biodata BLOB NOT NULL, PRIMARY KEY(id))

This table contains the following two BLOB columns.

A photo which contains employee picture.

Biodata file which contains employee details in file format.

```
MySQL  localhost:33060+ ssl  empdb  SQL > desc Employee;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name        | varchar(20) | NO   |     | NULL    |       |
| id          | int         | NO   | PRI | NULL    |       |
| salary      | float       | NO   |     | NULL    |       |
| Dept_id     | int         | NO   |     | NULL    |       |
| branch_name | varchar(20) | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.0471 sec)
```

The python_employee table is empty as of now let's insert employees' photo and bio-data file in it.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Insert Image and File as a BLOB data into MySQL Table : Let's insert employee photo and bio-data into a python_employee table. To insert BLOB data into MySQL Table from Python, you need to follow these simple steps :

- Install MySQL Connector Python using pip.
- Second, Establish MySQL database connection in Python.
- Create a function that can convert image and file into binary data.
- Then, Define the Insert query to enter binary data into the database table. All you need to know is the table its column details.
- Execute the INSERT query using cursor.execute(). In return, you should get some rows affected.
- After the successful execution of the query, commit your changes to the database.
- Close the Cursor and MySQL database connection.
- Most important, Catch SQL exceptions if any.
- At last, verify the result by selecting data from the MySQL table.

Let see the example now :

```
import mysql.connector
from mysql.connector import Error
def convertToBinaryData(filename):
    # Convert digital data to binary format
    with open(filename, 'rb') as file:
binaryData = file.read()
    return binaryData
def insertBLOB(emp_id, name, photo, biodataFile):
print("Inserting BLOB into Python_Employee table")
    try:
        connection = mysql.connector.connect(host='localhost',
                                             database='empdb',
                                             user='root',
                                             password='mysql20')
        cursor = connection.cursor()
sql_insert_blob_query = """ INSERT INTO Python_Employee
                          (id, name, photo, biodata) VALUES (%s,%s,%s,%s)"""
```

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

```
empPicture = convertToBinaryData(photo)
        file = convertToBinaryData(biodataFile)
        # Convert data into tuple format
insert_blob_tuple = (emp_id, name, empPicture, file)
        result = cursor.execute(sql_insert_blob_query, insert_blo _tuple)
connection.commit()
print("Image and file inserted successfully as a BLOB into python_employee table", result)
    except mysql.connector.Error as error:
print("Failed inserting BLOB data into MySQL table {}".format(error))
    finally:
        if (connection.is_connected()):
cursor.close()
connection.close()
print("MySQL connection is closed")
insertBLOB(101,"Suresh",r"C:\Users\user\Desktop\Biodata\Suresh\suresh_image.jpg",
r"C:\Users\user\Desktop\Biodata\Suresh\suresh_resume.pdf")
insertBLOB(102,"Venkatesh",r"C:\Users\user\Desktop\Biodata\Venkatesh\venkatesh_image.jpg",
r"C:\Users\user\Desktop\Biodata\Venkatesh\venkatesh_resume.pdf")
```

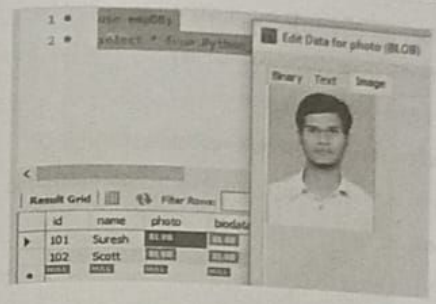
Output :

```
Inserting BLOB into Python_Employee table
Image and file inserted successfully as a BLOB into python_employee table None
MySQL connection is closed
Inserting BLOB into Python_Employee table
Image and file inserted successfully as a BLOB into python_employee table None
MySQL connection is closed
```

Let's have a look at Python_Employee table after inserting the image and file into it.

	id	name	photo	biodata
	101	Suresh	BLOB	BLOB
▶	102	Venkatesh	BLOB	BLOB

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL



6.7 FAMILIARIZE BREAD BOARD, RESISTOR, TRANSISTOR, DIODE, CAPACITOR, INDUCTOR, TRANSFORMER AND ADAPTOR COMPONENTS

Breadboard : The name breadboard comes from earlier days of electronic circuits, where people would literally use wooden boards (bread cutting boards) with screws or nails driven into them to make electronic connections. Let's see what a modern Breadboard is. Modern Breadboards are rectangular pieces of plastic with a Grid of holes that allow us to quickly and easily build electronic circuits by pushing electronic components into the holes.

Modern breadboards come in all sizes and shapes and in different colors. Some breadboards are transparent as well. The most common sizes are full-size breadboards, half-size breadboards, and mini breadboards. Also, most of the breadboards come with notches on the side that allows you to snap more than one board together.

Let's take a closer look at how a bread board works. See the below full size and half size breadboards and also a clear/transparent breadboard.

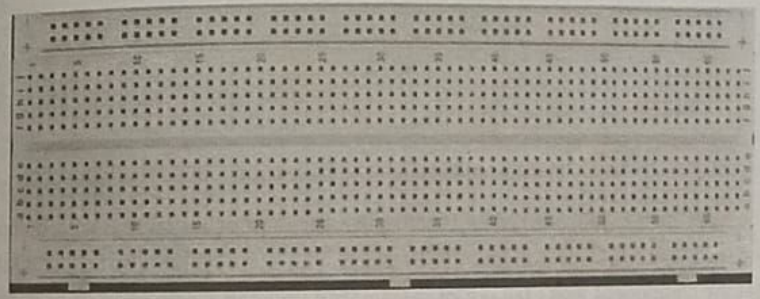


FIG 6.1 : Full-size Breadboard

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

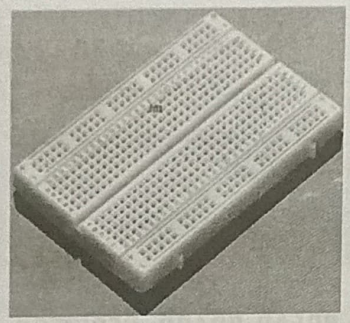


FIG 6.2 : Small-size Breadboard

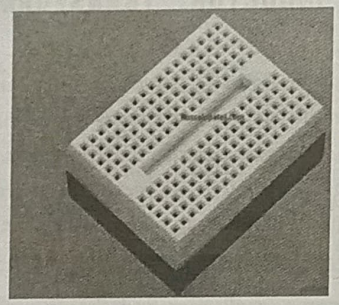


FIG 6.3 : Mini-size Breadboard

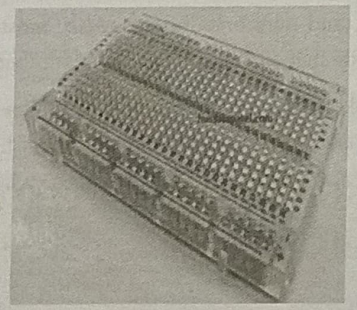


FIG 6.4 : Clear/Transparent Breadboard

The holes on a breadboard allow you to push the lead or metal legs of a component into them and tightly hold them into the place. This connection is strong enough such that the component won't fall out of its own, but you can easily snap in or snap out a component in case you want to change/replace it from that place.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Breadboard is also called as ***solderless breadboards*** (less commonly used name) – because you don't need to solder to bond the electronic components together.

Now, let's take a closer look at the below full-size breadboard.

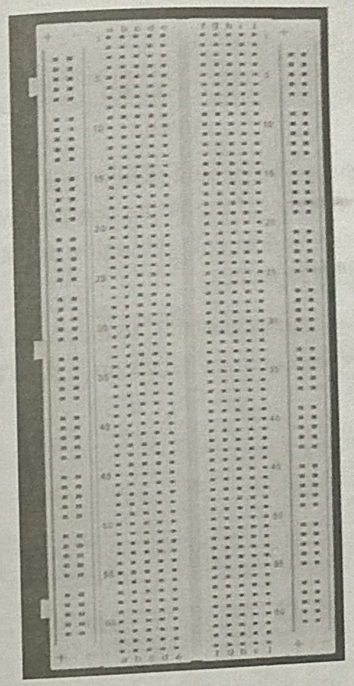


FIG 6.5 : Full-size breadboard

In the figure above, you see a full-sized breadboard. On either side of the breadboard are vertical lines (or) strips usually labeled with a 'red and black' (or) 'red and blue' lines and also having + (or) – sign. These lines are called ***busses*** (or) rails and are used to deliver power vertically to the entire circuit. Typically, the holes next to the red line (+) sign will connect to the positive battery terminal and the holes next to the blue line (–) sign will connect to the negative battery terminal.

The two columns on the inside of the breadboard work horizontally. What it means that internally they are wired horizontally and when connected to power, the power flows horizontally along the row of each column. So, if you look at row 1, the holes marked A, B, C, D, and E are connected and the holes in rows F, G, H, I, and J are connected (i.e. power will flow from A to E and F to J).

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

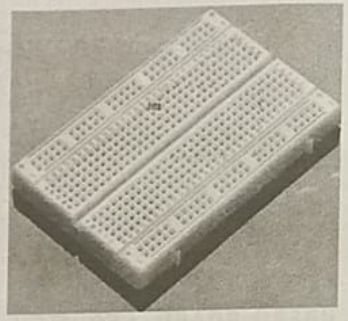


FIG 6.2 : Small-size Breadboard

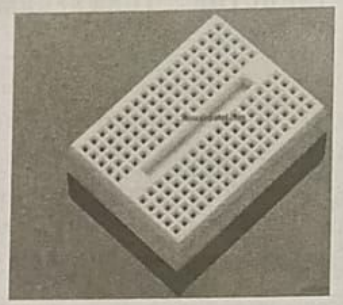


FIG 6.3 : Mini-size Breadboard

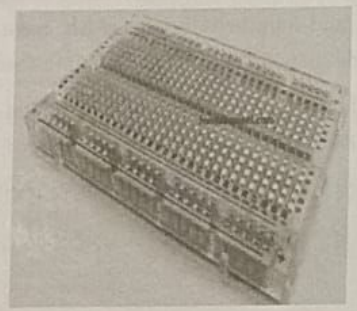


FIG 6.4 : Clear/Transparent Breadboard

The holes on a breadboard allow you to push the lead or metal legs of a component into them and tightly hold them into the place. This connection is strong enough such that the component won't fall out of its own, but you can easily snap in or snap out a component in case you want to change/replace it from that place.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Breadboard is also called as ***solderless breadboards*** (less commonly used name) – because you don't need to solder to bond the electronic components together.

Now, let's take a closer look at the below full-size breadboard.

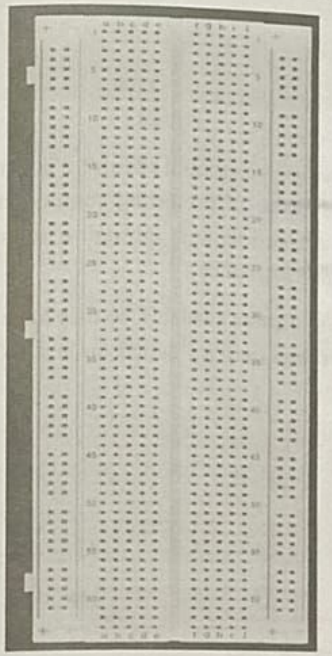


FIG 6.5 : Full-size breadboard

In the figure above, you see a full-sized breadboard. On either side of the breadboard are vertical lines (or) strips usually labeled with a 'red and black' (or) 'red and blue' lines and also having + (or) – sign. These lines are called ***busses*** (or) rails and are used to deliver power vertically to the entire circuit. Typically, the holes next to the red line (+) sign will connect to the positive battery terminal and the holes next to the blue line (–) sign will connect to the negative battery terminal.

The two columns on the inside of the breadboard work horizontally. What it means that internally they are wired horizontally and when connected to power, the power flows horizontally along the row of each column. So, if you look at row 1, the holes marked A, B, C, D, and E are connected and the holes in rows F, G, H, I, and J are connected (i.e. power will flow from A to E and F to J).

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

Resistor :

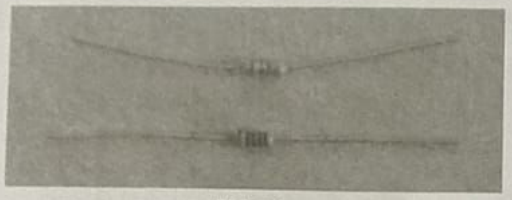


FIG 6.6 :

Resistors are used to change the amount of current flowing through a part of the circuit. This is often used as a means of protecting components which cannot handle large currents. The current allowed to flow through a resistor can be calculated using : I = V/R (current = voltage/resistance).

There are many different resistors available, so they have a sequence of coloured stripes to show their resistance. See here for details on how to decipher the colour coding.

Transistor :

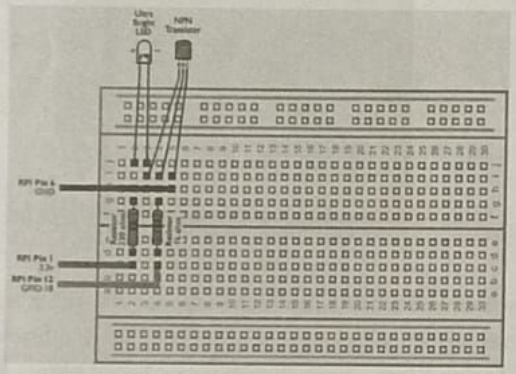


FIG 6.6 :

The Raspberry Pi's GPIO pins are only capable of supplying a low amount of power. If you try to power a component that requires a lot of power, you can cause permanent damage to the Raspberry Pi. Here, we aim to show you how to get around this issue by using a Transistor. Transistors can be used as a switch in a circuit, enabling you to control a higher power component without damaging your Raspberry Pi's GPIO pins. There are two types of transistors most commonly used. They are :

PNP Transistor :

- A PNP transistor conducts from collector to emitter.

- When you increase voltage to the base of a PNP transistor, the transistor is turned off more and more until it no longer conducts and completely shuts off.
- And as you decrease voltage to the base of a PNP transistor, the transistor turns on more and more, until the transistor fully conducts from collector to emitter.

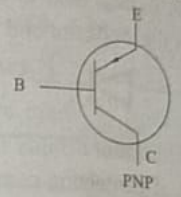


FIG 6.7 :

NPN Transistor :

- A NPN transistor conducts from emitter to collector.
- When you increase voltage to the base of a NPN transistor, the transistor is turned on more and more until it conducts fully from emitter to collector.
- And as you decrease voltage to the base of a NPN transistor, the transistor turns on less and less, until the voltage is so low, the transistor no longer conducts across emitter to collector and shuts off.

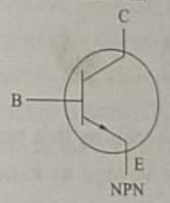


FIG 6.8 :

Diode : Diodes are simple electronic components that can visually resemble resistors. The main function of it is to direct the current flow one way, preventing it from returning. The use in electronic circuits could involve protecting other elements, assuring the one-way flow of the electrons and regulating your voltage. Because of this function diodes have to be oriented correctly, otherwise, it will stop current from flowing. It is worth mentioning that LED lights are also classed as diodes, because of their qualities and structure.

Schematics : Diodes are marked on the schematics like this and the triangle indicates the direction of the current (note that electron flow is opposite).

The Pi can support 100 Kbits standard mode as well as 400 Kbits "fast mode". The higher speeds depending on cable length, board model and software processing.

With its 7 bit addressing, I2C can support up to 127 devices (or nodes). The two lines are called **SCL** and **SDA**. SCL is the clock line for synchronizing transmission. SDA is the data line through which bits of data are sent (or) received. During transmission, the first byte includes the 7 bit address plus a read/write bit. Subsequent bits represent the actual data.

I2C connection to the RPi is made using GPIO board pins 3 for SDA and 5 for SCL (BCM mode GPIO 2 and GPIO 3). The RPi GPIO operates at 3.3v so care must be taken to ensure connections to slave devices are also 3.3v. I2C wiring distance is considered relatively short, typically from inches to a few meters. Distance is affected by data speed, wire quality and external noise.

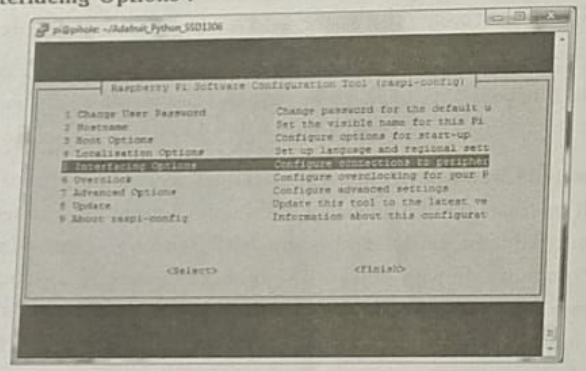
Configuring I2C : I2C is a very commonly used standard designed to allow one chip to talk to another. So, since the Raspberry Pi can talk I2C we can connect it to a variety of I2C capable chips and modules.

The I2C bus allows multiple devices to be connected to your Raspberry Pi, each with a unique address that can often be set by changing jumper settings on the module. It is very useful to be able to see which devices are connected to your Pi as a way of making sure everything is working.

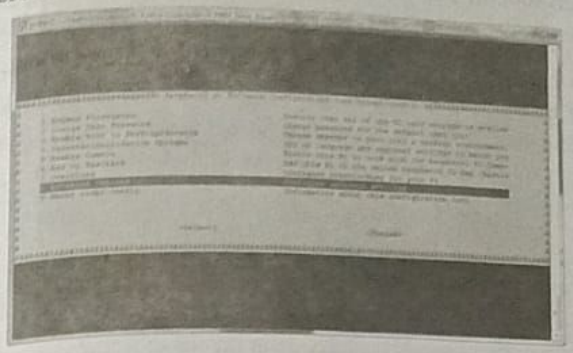
```
sudo apt-get install -y python-smbus
sudo apt-get install -y i2c-tools
```

Installing Kernel Support (with Raspi-Config) : Run **sudoraspi-config** and follow the prompts to install i2c support for the ARM core and linuxkernel.

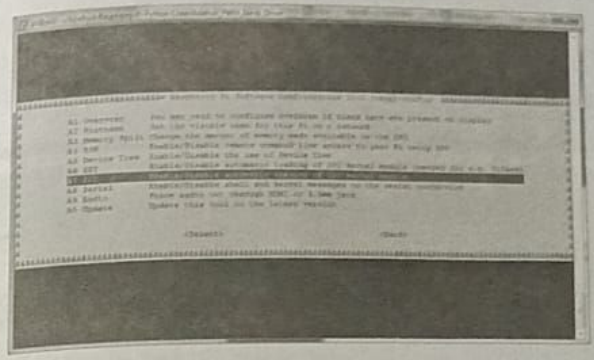
Go to Interfacing Options :



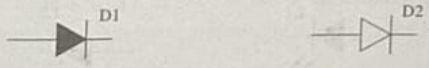
On older versions, look under Advanced :



Then select I2C :



Enable! :



Bands and Forward Voltage and PIV : The direction of the flow is indicated on a diode with a band. The band is placed next to the **cathode (–)** pin, which means voltage (+) should be applied on the opposite end.

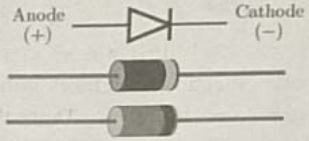


FIG 6.9 :

When selecting a diode, one of the most important value is Peak Inverse Voltage. This tells us max V that can be applied to the diode mounted in reverse (stopping the current) before diode will receive damage and allow that current to travel through. This information is provided on data sheets from manufacturers.

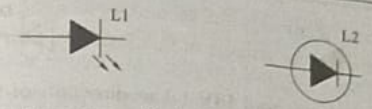
Zener and LED diodes

There are another 2 types of diodes.

Zener Diode :

These work in a slightly opposite manner to a normal diode, connected the same way as a regular diode, it drops the voltage as expected 0.7V (for a silicone one), and however, connected in a reversed manner, it will stop the current until the Voltage reaches the Zener Voltage. At this point, no matter how much current (I) you will run through this (until limit), it will limit the same amount of Voltage. i.e., diode with 6V Zener Voltage attached to a 12V (through a resistor) would lower your voltage to 6V and keep it there regardless of the current supplied (up to the diode max current limit). In the same situation should voltage drop below 6V, the diode will stop conducting electricity, as the diode with 6V Zener Voltage needs 6V+ to be conductive.

LED : Apart from the obvious light emission, LED also acts as diodes and if the voltage is low (up to 5V) we can see the zener diode qualities. Bear in mind that operating current of the LED is very low and the 'zener' voltage of the LED varies per colour (this also influences the forwarding Voltage of the LED, as it is different for each colour).



LEDs are a very effective source of light, require low voltage (3-12V range) and usually current under 20mA. To as other diodes – to make LED shine, the forward current value must be met.

Capacitor : Capacitors are made to store an electric charge, comparison to a rechargeable battery is often mentioned. It takes some time to charge the capacitor, and then the energy is stored until its release. That's the very basic way of describing what it does. Generally, it takes the same amount of time to charge a capacitor and to discharge it. When the component is being charged, the voltage starts to increase, a spike in voltage is not linear, and the more energy is stored, the slower voltage (and charge) increases.

Capacitors respond differently to the DC and AC power. When DC is applied to the capacitor in series (coupling), it will charge up, and upon reaching its capacitance no current will be passed through (see leakage below though). If AC is applied, the capacitor will attempt to even out the voltage spikes by releasing stored charge.

The capacity of the component is expressed in farads (F), other characteristic include: maximum voltage we can drop across the pins, leakage – which tells us how much current can go through the component causing the capacitor to drain it. Resistance especially in series (ESR) this value is generally small, but can add up when multiple capacitors are used, and tolerance which like in resistors tells us how precise is the capacitor's specification.

Transformer : The transformer is an electrostatic device which is used to transfer electrical energy (voltage or current) from one circuit to another by mutual induction of two electric circuits without change in frequency, which is working under the principle of electromagnetic induction.

Principle of operation :

- The transformer is working under the principle of electromagnetic induction.
- By using this principle, which transfers electrical energy from one winding to another winding by mutual induction between the two windings.
- An alternating flux is established in the magnetic core when the primary winding is energized from an ac source (V1) and the secondary is open circuited.

- This flux links both the primary and secondary windings; thereby an emf is induced in them due to the rate of change of flux linkages with the windings.

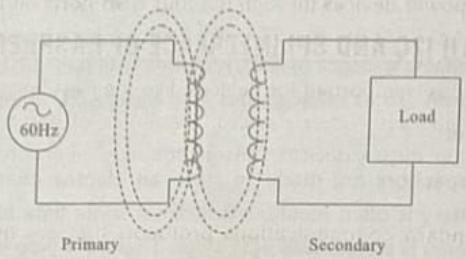


FIG 6.10 :

Power Adaptor :

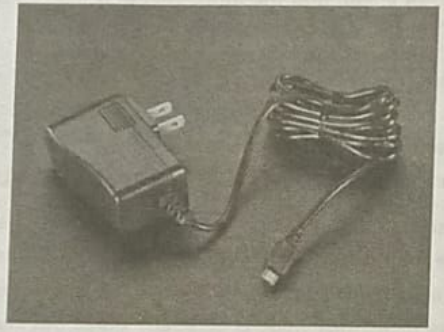


FIG 6.11 :

Official Raspberry Pi power supply, ideal for use with any Raspberry Pi board. This power supply comes with 1.5 m Micro USB lead. It supports up to 2.5A of current which is plenty to power devices through the four USB ports on the board.

6.8 WORK WITH I2C AND SPI INTERFACE OF RASPBERRY PI

Data communications is important for devices, like the Raspberry Pi, to communicate and exchange "data" with other devices. Examples of devices that the RPi may communicate include: display devices, sensors, robotics, other computers, input devices, industrial controls, scientific instruments etc.,

All represent standard communications protocols that are available through the Raspberry Pi GPIO (General Purpose Input/Output) pins. Each has characteristics that may be better for a particular project. Here we will discuss two most important communication protocols Raspberry Pi I2C and SPI.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Official Raspberry Pi power supply, ideal for use with any Raspberry Pi board. This power supply comes with 1.5m Micro USB lead. It supports up to 2.5A of current which is plenty to power devices through the four USB ports on the board.

6.8 WORK WITH I2C AND SPI INTERFACE OF RASPBERRY PI

Data communications is important for devices, like the Raspberry Pi, to communicate and exchange "data" with other devices. Examples of devices that the RPi may communicate include: display devices, sensors, robotics, other computers, input devices, industrial controls, scientific instruments etc.,

All represent standard communications protocols that are available through the Raspberry Pi GPIO (General Purpose Input/Output) pins. Each has characteristics that may be better for a particular project. Here we will discuss two most important communication protocols Raspberry Pi I2C and SPI.

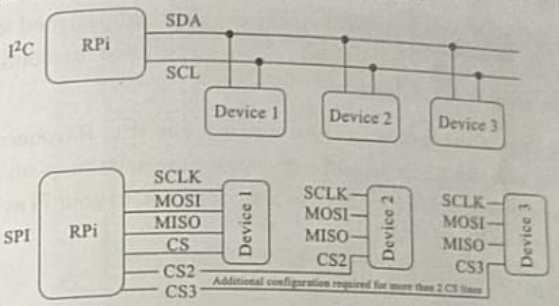


FIG 6.12 :

Table of RPi Serial Communications Methods via GPIO :

Name	Description	Function
I²C	Inter-Integrated circuit	Half duplex, serial data transmission used for short-distance between boards, modules and peripherals. Uses 2 pins.
SPI	Serial Peripheral Interface bus	Full-duplex, serial data transmission used for short-distance between devices. Uses 4 pins.

Inter Integrated Circuit (I2C) : I2C is bidirectional, synchronous, serial communications interface. It operates on two lines in a half-duplex mode. It was originally created by Philips Semiconductor which later became NXP Semiconductors. A single master (the RPi) can communicate with one (or) more slave devices. Each connected device is selected via a 7 bit address (10 or more bit addressing is possible, but more complex). Originally limited to 100 Kbits per second, it now supports faster transmission rates.

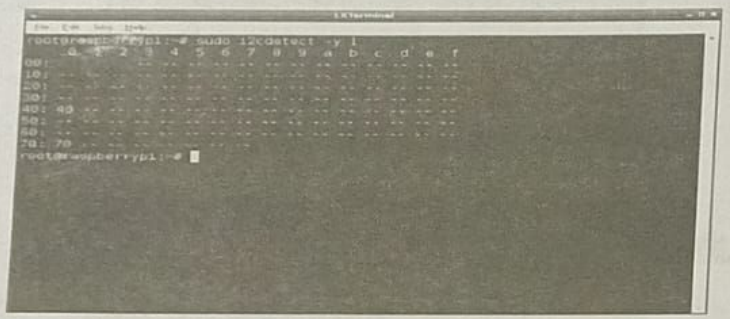
XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

```
sudo reboot
```

Testing I2C

Now when you log in you can type the following command to see all the connected devices.

```
sudo i2cdetect -y 1
```



This shows that two I2C addresses are in use – 0x40 and 0x70.

These values will be different for you depending on what is currently attached to the I2C pins of your Raspberry Pi. Note that if you are using one of the very first Raspberry Pis (a 256MB Raspberry Pi Model B) then you will need to change the command to:

```
sudo i2cdetect -y 0
```

The Raspberry Pi designers swapped over I2C ports between board releases. Just remember : 512M Pi's use i2c port 1, 256M ones use i2c port 0.

When you are finished in raspi-config, reboot for the i2c modules to automatically load into the kernel.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Serial Peripheral Interface (SPI) : SPI (Serial Peripheral Interface) is a bidirectional, synchronous, serial communications interface - like I2C. Also like I2C, it is used for relatively short distances. Unlike I2C, however, SPI operates at full duplex, meaning data can be sent and received simultaneously. Additionally, it can operate at faster data transmission rates, with rates upwards of 8 Mbits (or) more possible on the RPi.

SPI can communicate with multiple devices through two ways. The first is by selecting each device with a Chip Select line. A separate Chip Select line is required for each device. This is the most common way RPi's currently use SPI. The second is through daisy chaining where each device is connected to the other through its data out to the data in line of the next.

There is no defined limit to the number of SPI devices that can be connected. However, practical limits exist due to constrains by the number of hardware select lines available on the master in the first method (or) the complexity of passing data through devices in the second daisy chain method.

The RPi has two Chip Select (CE0 and CE1) lines readily available. More can be defined by configuring other GPIO pins and through software programming.

Default RPi GPIO Pins used for SPI :

S.No.	Board Pin	BCM Number	Function
1.	19	GPIO 10	MOSI - Master Out Slave In
2.	21	GPIO 9	MISO - Master In Slave Out
3.	23	GPIO 11	SCLK - Serial Clock
4.	24	GPIO 8	CE0 - Chip Select 0
5.	26	GPIO 7	CE1 - Chip Select 1

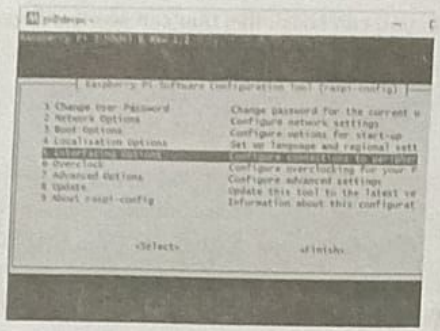
Configuring SPI

Installing Kernel Support (with Raspi-Config).

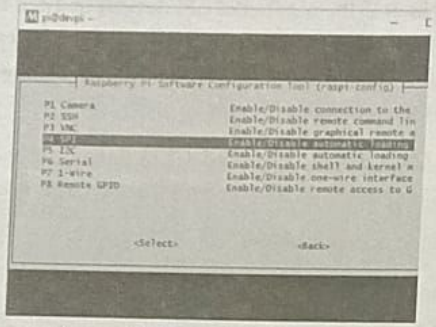
Run **sudoraspi-config** and follow the prompts to install i2c support for the ARM core and linuxkernel.

Go to **Interfacing Options.**

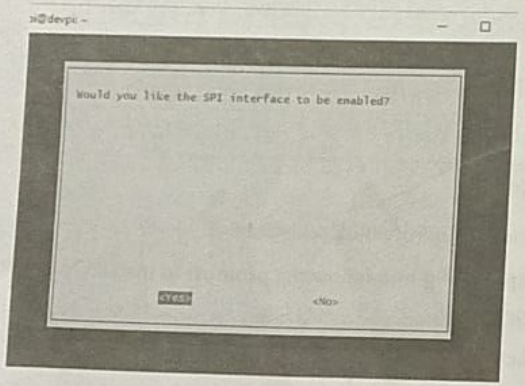
XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING



Then select SPI :



When asked if you want to **enable** select **YES**



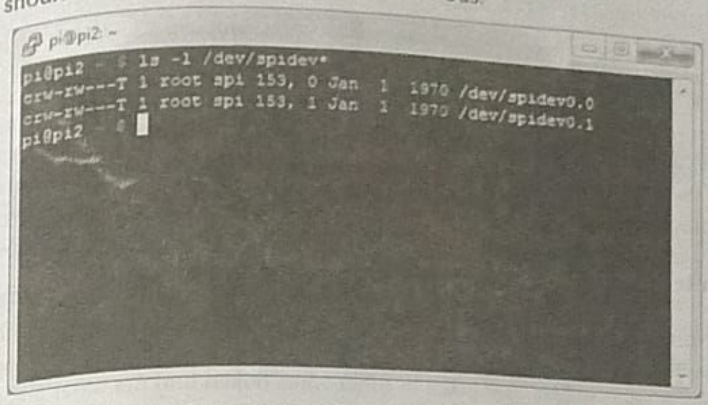
Now reboot your Pi to make the SPI interface appear.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Next time you log in you can check that you can see the devices with

```
ls -l /dev/spidev*
```

you should see two 'devices' one for each SPI bus.



When you are finished in raspi-config, reboot for the SPI modules to automatically load into the kernel.

6.9 TURN ON AND OFF LED WITH RASPBERRY PI USING PYTHON PROGRAM

One of the important parts of Raspberry Pi is its GPIO (or) General Purpose Input/Output ports. They are the little pins sticking out of the circuit board and allow you to plug various devices into your Raspberry Pi. With a little programming, you can then control them or detect what they are doing. Here we are going to show you how to light an LED. In addition to your Raspberry Pi running Raspbian, the other components required are :

1. A Breadboard.
2. An LED.
3. A resistor(anything from 220 Ohm to 1k Ohm).
4. Two Male-Female jumper wires.

1. **The Breadboard :** The breadboard is a way of connecting electronic components to each other without having to solder them together. They are often used to test a circuit design before creating a Printed Circuit Board (PCB).

   The holes on the breadboard are connected in a pattern.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

pressed, (or) temperature, (or) light. The diagram below left shows the pin layout for a Raspberry Pi Models A and B (Rev 2 - the original Rev 1 Pi is slightly different), looking at the Raspberry Pi with the pins in the top right corner. The new 40 pin Raspberry Pi's shares exactly the same layout of pins for the top 13 rows of GPIO pins.

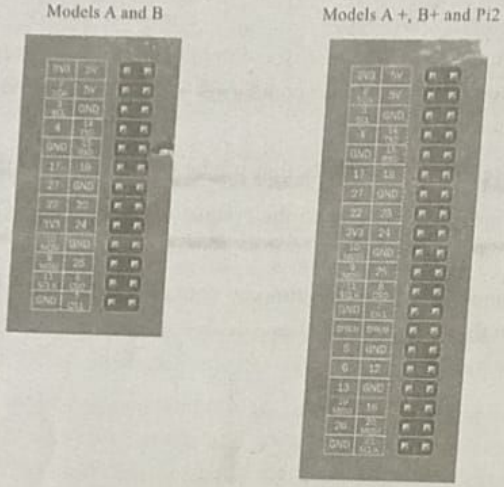


FIG 6.15 :

Building the Circuit : The circuit consists of a power supply (the Raspberry Pi), an LED that lights when the power is applied and a resistor to limit the current that can flow through the circuit.

You will be using one of the 'ground' (GND) pins to act like the 'negative' (or) 0 volt ends of a battery. The 'positive' end of the battery will be provided by a GPIO pin. Here we will be using pin 18. When they are 'taken high', which means it outputs 3.3 volts, the LED will light. Now take a look at the circuit diagram below.

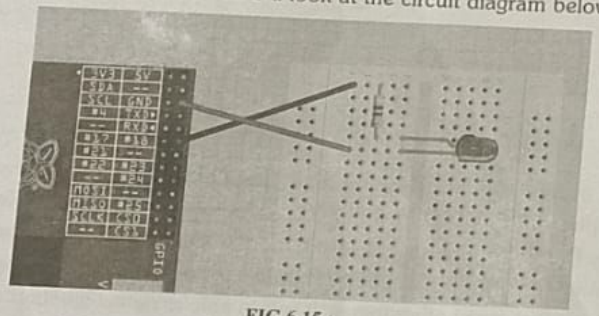


FIG 6.15 :

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

You should turn your Raspberry Pi off for the next bit, just in case you accidentally short something out.

- Use one of the jumper wires to connect a ground pin to the rail, marked with blue, on the breadboard. The female end goes on the Raspberry Pi's pin and the male end goes into a hole on the breadboard.
- Then connect the resistor from the same row on the breadboard to a column on the breadboard, as shown above.
- Next, push the LEDs legs into the breadboard, with the long leg (with the kink) on the right.
- Lastly, complete the circuit by connecting pin 18 to the right hand leg of the LED. This is shown here with the orange wire.

The Code : You are now ready to write some code to switch the LED on. Turn on your Raspberry Pi Open Terminal and Launch IDLE IDE by typing

sudo idle

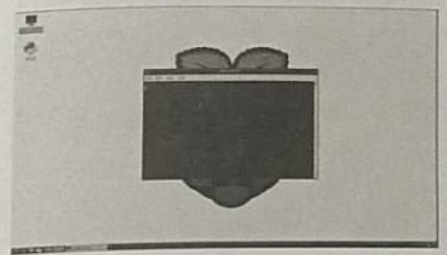
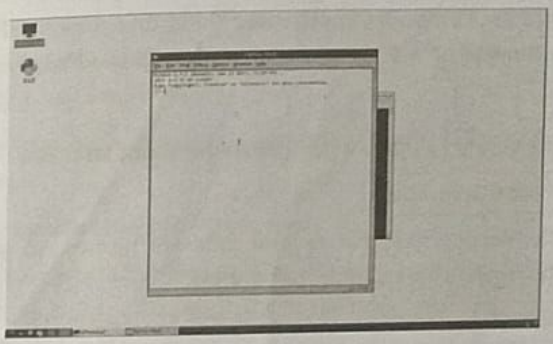


FIG 6.16 :

This launches IDLE with superuser privileges which is necessary execute scripts for controlling the GPIO pins.



XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

- Buzzer/piezo speaker.
- And of course any Raspberry Pi board with Raspbian OS installed.

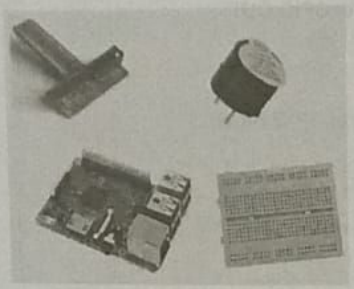


FIG 6.18 :

Step 2 : The Circuit : The connections are pretty easy, see the image below with breadboard circuit and connect the buzzer to the Raspberry pi in the same manner.

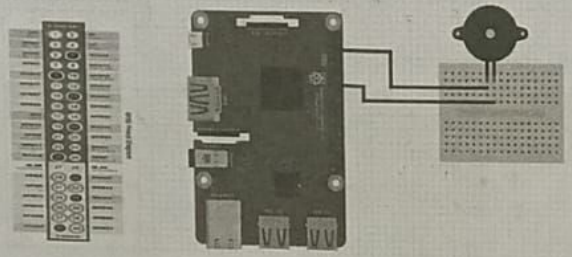


FIG 6.19 :

Step 3 : Python Code : In the program below, the first thing you do is to import the library for GPIO and sleep. The next step is to initialize pin 23 as an output pin with GPIO.setup( ) function. The While True loop runs over and over again, forever. In the main loop, you make a beep sound with GPIO.output( ) function and "pause" the program for 1 second with sleep( ) function.

Write the code in Python IDE and run it.

```
#Libraries
import RPi.GPIO as GPIO
from time import sleep
#Disable warnings (optional)
GPIO.setwarnings(False)
#Select GPIO mode
GPIO.setmode(GPIO.BCM)
```



```
#Set buzzer - pin 23 as output
buzzer=23
GPIO.setup(buzzer,GPIO.OUT)
#Run forever loop
while True:
GPIO.output(buzzer,GPIO.HIGH)
   print("Beep")
sleep(1) # Delay in seconds
GPIO.output(buzzer,GPIO.LOW)
print("No Beep")
sleep(1)
```

Once you execute the above code buzzer will make a beep sound continuously.

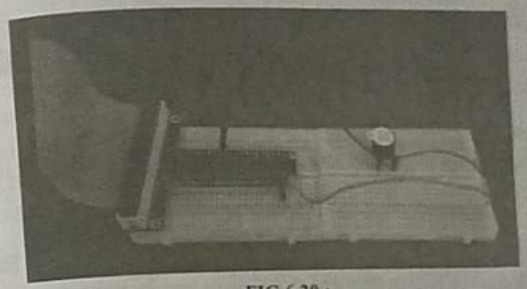


FIG 6.20 :

6.11 CONNECT TO WIRED (OR) WIRELESS NETWORK WITH RASPBERRY PI

Connecting to a Wired Network : First, if you have a Raspberry Pi model A, there is no RJ45 connector for Ethernet. In this case, your best option for Internet access is to use a wireless USB adaptor.

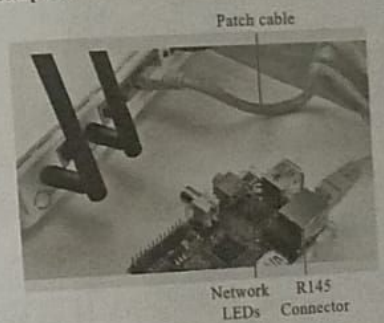
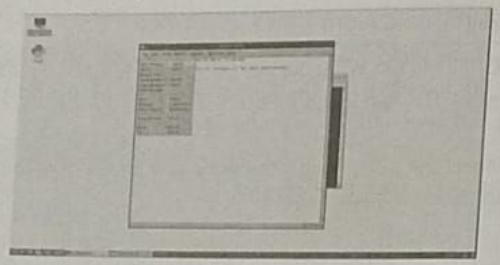


FIG 6.21 :

After the IDLE launches, open a new window by FILE>OPEN (or) Ctrl+N.



Type the code below in the window.

```
import time
import RPi.GPIO as GPIO    ##Import GPIO library
GPIO.setmode(GPIO.BCM)  ##Use board pin numbering
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT) ## Setup GPIO Pin 18 to OUT
while True:
    print("LED on")
    GPIO.output(18,GPIO.HIGH) ##Turn on Led
    time.sleep(1)     ##Wait for one second
    print("LED off")
    GPIO.output(18,GPIO.LOW) ##Turn off Led
    time.sleep(1)  ##Wait for one second
```

Explanation :

`import RPi.GPIO as GPIO`

The first line tells the Python interpreter (the thing that runs the Python code) that it will be using a library that will tell it how to work with the Raspberry Pis GPIO pins.

`import time`

Imports the time library so that we can pause the script later on.

`GPIO.setmode(GPIO.BCM)`

Each pin on the Raspberry Pi has several different names, so you need to tell the program which naming convention is to be used.

`print("LED on")`

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

This line prints some information to the terminal.

`GPIO.output(18, GPIO.HIGH)`

This turns the GPIO pin on. What this actually means is that the pin is made to provide power of 3.3volts. This is enough to turn the LED in our circuit on.

`time.sleep(1)`

Pauses the Python program for 1 second

`print("LED off")`

This line prints some information to the terminal.

`GPIO.output(18, GPIO.LOW)`

This turns the GPIO pin off, meaning that the pin is no longer supplying any power.

Save the above code by FILE>SAVE or Ctrl+S. To run your code RUN>RUN (or) Ctrl+F5.

The Led will start blinking now :

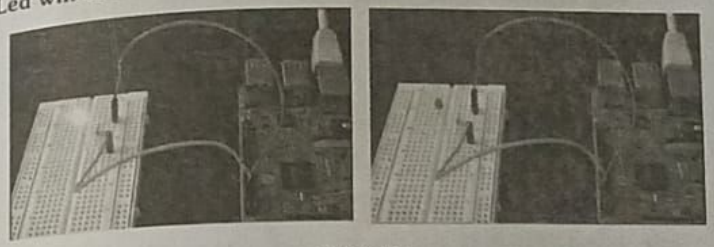


FIG 6.17 :

6.10 MAKE A BUZZING SOUND WITH RASPBERRY PI USING PYTHON PROGRAM

Here we will discuss how to use a buzzer (or) piezo speaker with Raspberry Pi. Buzzers can be found in alarm devices, computers, timers and confirmation of user input such as a mouse click (or) keystroke.

The process of making a sound through buzzers using Raspberry pi can be explained using three steps.

Step 1 : What you need – Hardware.

The following hardware components are required for doing the experiment.

- GPIO Breakout(optional).
- Breadboard.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

FIG 6.24 :

5. Select your Wi-Fi Network and connect to it
   (a) This is where you will need to switch the mouse out with the keyboard so that you can enter your Wi-Fi networks password.
   (b) Once done, switch back to the mouse.
6. Once connected to your home Wi-Fi Network, hover over the Wi-Fi symbol and view the IP address (general IP addresses look like this, 192.168.xx.xxx) of the Raspberry Pi.

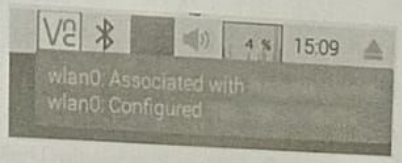


FIG 6.25 :

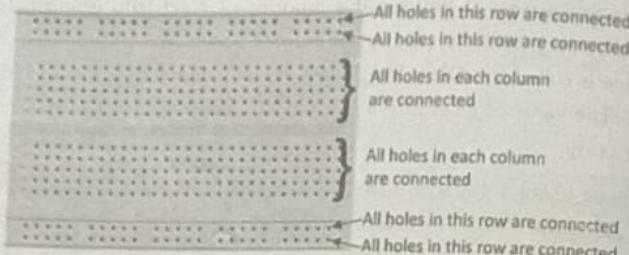
Write down the IP address of the Raspberry Pi, it will be required so that you can remotely connect to it.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

OBJECTIVE TYPE QUESTIONS

1. The Raspberry Pi is defined as the _____. [ ]
   (a) Micro Computer (b) Mega Computer
   (c) Minicomputer (d) Nano Computer
2. Raspberry Pi consists of a _____ quad-core processor (or) microprocessor. [ ]
   (a) 16-bit (b) 32-bit
   (c) 64-bit (d) 128-bit
3. What are the capabilities of raspberry Pi? [ ]
   (a) Browsing the internet (b) Making spreadsheets
   (c) Word pressing (d) All of the above
4. Which operating system Raspberry Pi has? [ ]
   (a) Linux (b) OpenBSD
   (c) NetBSD (d) All of the above
5. Does Raspberry Pi need external hardware? [ ]
   (a) TRUE (b) FALSE
   (c) Can be true (or) false (d) Cannot say
6. How power supply is done to RPi? [ ]
   (a) USB connection (b) Internal battery
   (c) Charger (d) Adapter
7. In oracle database variable length column is declared by _____. [ ]
   (a) Varchar (b) Varchar 3
   (c) Varchar 2 (d) None of the mentioned
8. Character data can be stored as _____. [ ]
   (a) Fixed length string (b) Variable length string
   (c) Either fixed (or) variable length string (d) None of the mentioned
9. What do we use to connect TV to RPi? [ ]
   (a) Male HDMI (b) Female HDMI
   (c) Male HDMI and Adapter (d) Female HDMI and Adapter

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING



With the above breadboard, the top rows of holes are all connected together – marked with blue dots. And so are the second row of holes – marked with red dots. The same goes for the two rows of holes at the bottom of the breadboard.

In the middle, the columns of wires are connected together with a break in the middle. So, for example, all the green holes marked are connected together, but they are not connected to the yellow holes, nor the purple ones. Therefore, any wire you poke into the green holes will be connected to other wires poked into the other green holes.

2. **The LED :** When you pick up the LED, you will notice that one leg is longer than the other. The longer leg (known as the **'anode'**) is always connected to the positive supply of the circuit. The shorter leg (known as the ***'cathode'***) is connected to the negative side of the power supply, known as ***'ground'***. LED stands for Light Emitting Diode, and glows when electricity is passed through it.

   LEDs will only work if power is supplied the correct way round (i.e., if the 'polarity' is correct). You will not break the LEDs if you connect them the wrong way round – they will just not light. If you find that they do not light in your circuit, it may be because they have been connected the wrong way round.

FIG 6.13 :

3. **The Resistor :** You must always use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi can only supply a small current (about 60mA). The LEDs want to draw more current, and if allowed they will burn out the Raspberry Pi. Therefore putting the resistors in the circuit will ensure that only this small current will flow and the Raspberry Pi will not be damaged.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of current that is allowed to flow. The measure of resistance is called the **Ohm (&!)** and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

You will be using a 330&! resistor. You can identify the 330&! resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied :

- If there are four colour bands, they will be Orange, Orange, Brown and then Gold.
- If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.

It does not matter which way round you connect the resistors. Current flows in both ways through them.

FIG 6.14 :

4. **Jumper Wires :** Jumper wires are used on breadboards to 'jump' from one connection to another. The ones you will be using in this circuit have different connectors on each end. The end with the 'pin' will go into the Breadboard. The end with the piece of plastic with a hole in it will go onto the Raspberry Pi's GPIO pins.

FIG 6.15 :

The Raspberry Pi's GPIO Pins : GPIO stands for General Purpose Input Output. It is a way the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits. The Raspberry Pi is able to control LEDs, turning them on (or) off (or) motors, (or) many other things. It is also able to detect whether a switch has been

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

If you have a Raspberry Pi model B, plug an Ethernet patch cable into its RJ45 socket and then connect the other end to a spare socket on the back of your home hub/router.

The network LEDs on your Raspberry Pi should immediately start to flicker as the Raspberry Pi connects to your network.

FIG 6.22 :

Raspbian distributions are preconfigured to connect to any network using DHCP (Dynamic Host Configuration Protocol). They will automatically be assigned an IP address as long as DHCP is enabled on your network.

If the network LEDs on your Raspberry Pi do not light up when you plug it into the home hub, check that you have not used the Uplink RJ45 socket on the hub or try a different cable.

If the LEDs blink, but you cannot connect to the Internet on your Raspberry Pi using a browser, check that DHCP is enabled on your network management console. Look for an option like that shown in the Fig. 6.23.

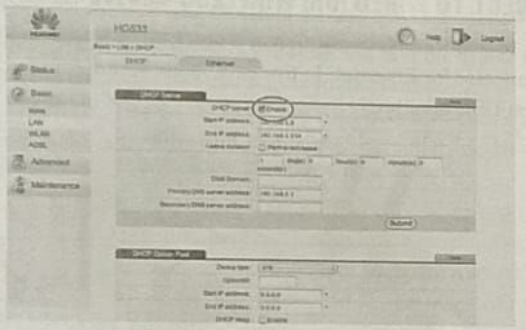


FIG 6.23 :

Setting Up a Wireless Connection : These instructions are for setting up your Raspberry Pi on your home network and to prepare it for remote connection.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

Setting up a wireless connection is very easy if you are using a recent version of the Raspbian distribution, as this includes a WiFi Config utility with a shortcut on the desktop. If you are not using a recent distribution, then you should update to new one.

Simply plug a compatible USB wireless adapter (most are) into one of the USB sockets on your Raspberry Pi, launch the WiFi Config utility and then click the Scan button to search for access points. Double-click on the access point (for your home hub) that you want to join and then enter the password in the PSK field. Finally, click Connect to join the network.

Components needed are :

- Raspberry Pi.
- Raspberry Pi Power Adapter.
- Mini HDMI Adapter For Pi-Zeros (comes with your Raspberry Pi-Zero).
- USB wireless Adapter For Pi-Zeros (comes with your Raspberry Pi-Zero).
- HDMI cable.
- Monitor (or) TV that has a HDMI port.
- Keyboard.
- Mouse.

Setup Wi-Fi Connection :

1. Connect your Raspberry Pi to a TV (or) Monitor with a HDMI port.
2. Connect a mouse to your Raspberry Pi using the USB adapter.
   (a) Due to the single port of the USB adapter, you need to switch between the mouse and the keyboard.
   (b) If you are using a monitor with USB ports on it, it is recommended that you plug both keyboard and mouse into the monitor and then use a USB cable with a USB-b connector end (the one that looks like a square), to connect the monitor to the Raspberry Pi via USB adapter. This will eliminate the need to switch between the keyboard and mouse using the single port USB adapter.
3. Connect the power adapter to the Raspberry Pi and plug it in.
4. Once the Raspberry Pi has booted to the desktop go to the Network connection symbol in the taskbar at the top right of the screen and click on it.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING

An event loop is basically telling the code to keep displaying the window until we manually close it. It runs in an infinite loop in the back-end.

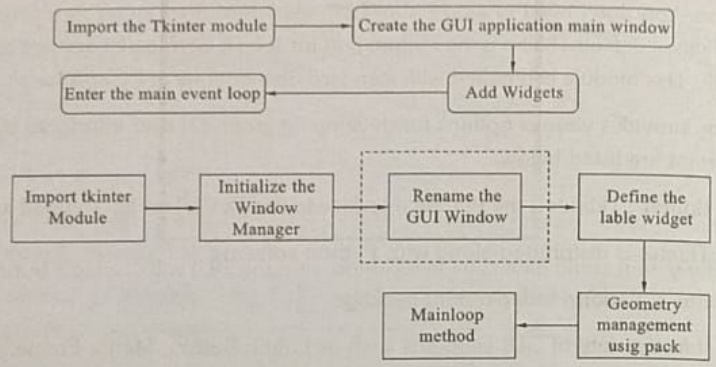


FIG 5.1 :

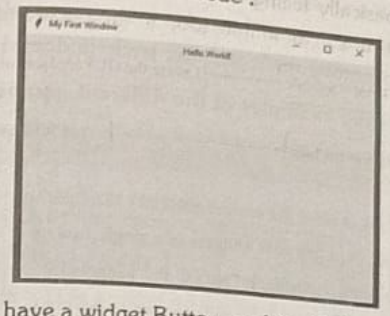
EXAMPLE

```
import tkinter
window = tkinter.Tk()
window.title("My First Window") # to rename the title of the window
# Code to add widgets will go here...
label = tkinter.Label(window, text = "Hello World!").pack() # pack is used to show the object in the window
window.mainloop()
```

- First, you import the key component, i.e., the tkinter module.
- As a next step, you initialize the window manager with the tkinter.Tk() method and assign it to a variable. This method creates a blank window with close, maximize and minimize buttons on the top as a usual GUI should have.
- Then as an optional step, you will rename the title of the window as you like with window.title(title_of_the_window).
- Next, you make use of a widget called ***Label,*** which is used to insert some text into the window.
- Then, you make use of Tkinter's geometry management attribute called **pack()** to display the widget in size it requires.
- Finally, as the last step, you use the mainloop() method to display the window until you manually close it. It runs an infinite loop in the backend.

WARNING XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL

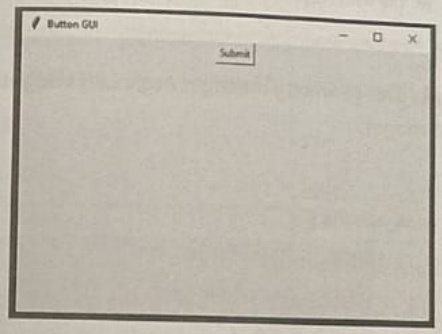
Check out the output for the above code :



Similarly, you could have a widget Button, and the GUI will display a button instead of some text (Label).

```
import tkinter
window = tkinter.Tk()
window.title("Button GUI")
button_widget = tkinter.Button(window,text="Submit")
button_widget.pack()
tkinter.mainloop()
```

Check out the output for the above code :



5.2 DESIGN GUI USING DIFFERENT GEOMETRY MANAGERS

Just instantiating (creating) a widget does not necessarily mean that it will appear on the screen. To get the widget to appear, we need to tell the parent widget where to put it. To do that, we use one of Tkinter's three geometry managers (also known as ***layout managers***). A geometry manager is some code that runs on the backend of Tkinter (we don't interact with the geometry managers directly). We simply choose which geometry manager we want to use and give it some parameters to work with.

XEROX / PHOTOCOPYING OF THIS BOOK IS ILLEGAL WARNING