

Building Modern GUIs with tkinter and Python

Building user-friendly GUI applications with ease



Saurabh Chandrakar

Dr. Nilesh Bhaskarrao Bahadure

bpb



Building Modern GUIs

— with —

tkinter and Python

Building user-friendly GUI applications with ease



Saurabh Chandrakar

Dr. Nilesh Bhaskarrao Bahadure

bpb

Building Modern GUIs with tkinter and Python

Building user-friendly GUI applications with ease

**Saurabh Chandrakar
Dr. Nilesh Bhaskarrao Bahadure**



www.bpbonline.com

Copyright © 2023 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

First published: 2023

Published by BPB Online

WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-569

www.bpbonline.com

Dedicated to

My Parents

Dr Surendra Kumar Chandrakar
and

Smt. Bhuneshwari Chandrakar

Brother Shri Pranav Chandrakar

to my wife Priyanka Chandrakar

and to my lovely son Yathartha Chandrakar

- *Saurabh Chandrakar*

My Parents

Smt. Kamal B. Bahadure

and

Late Bhaskarrao M. Bahadure

to my in-laws

Smt. Saroj R. Lokhande and Shri. Ravikant A. Lokhande

and to my wife Shilpa N. Bahadure

and to beautiful daughters Nishita and Mrunmayee

And to all my beloved students.

- *Dr. Nilesh Bhaskarrao Bahadure*

About the Authors

- **Saurabh Chandrakar** is a Research & Development Engineer (Dy. Manager) at Bharat Heavy Electricals Limited (BHEL) Hyderabad. He is the winner of the best executive award on Operations Division by BHEL Hyderabad. Recently, he has been awarded the prestigious BHEL Excellence Award under Anusandhan category for Redundant Composite Monitoring System of Power Transformers project. He has 20 copyrights and 1 patent granted. Additionally, he has 6 patents filed. Moreover, he has published 3 books in reputed publications such as BPB New Delhi (Programming Techniques using Python), Scitech Publications Chennai (Programming Techniques using matlab) and IK International publishers (Microcontrollers and Embedded System Design). He has also launched 1 video course on BPB titled “First Time Play with Basic, Advanced Python concepts and complete guide for different python certification exams all in one umbrella.”
- **Nilesh Bhaskarao Bahadure** received his Bachelor of Engineering degree in Electronics Engineering in 2000, his Master of Engineering degree in Digital Electronics in 2005, and the Ph.D. degree in Electronics in 2017 from KIIT Deemed to be University, Bhubaneswar, India. He is currently an Associate Professor in the Department of Computer Science and Engineering at Symbiosis Institute of Technology (SIT), Nagpur, Symbiosis International (Deemed University) (SIU), Pune, Maharashtra, India. He has more than 20 years of experience. Dr. Bahadure is a life member of IE(I), IETE, ISTE, ISCA, SESI, ISRS, and IAENG professional organizations. He has published more than 40 articles in reputed international journals and conferences, and has 5 books to his credit. He is the reviewer of many indexed journals such as IEEE Access,

IET, Springer, Elsevier, Wiley and so on. His research interests are in the areas of Sensor Technology, the Internet of Things, and Biomedical Image Processing.

About the Reviewer

Dr. Prasenjeet Damodar Patil received B.E in E&TC Engineering from Sant Gadgebaba Amravati University and M. Tech. from Walchand College of Engineering Sangli, India. He did his Ph.D. degree in E&TC Engineering from Sant Gadgebaba Amravati University. He has 14+ years of teaching experience. Currently, he is working as Associate Professor at School of Computing, M.I.T A.D.T University, Pune. He has published more than 15 papers in reputed Journals. His research interest includes Computational Electromagnetics applications in Integrated Optics, IoT & Digital Image Processing.

Acknowledgements

- First and foremost, I would like to thank you all for selecting this book. It has been written with the beginner reader in mind. First of all, I take this opportunity to greet and thank my mentor Prof. Nilesh Bahadure Sir for motivating me and always communicating his expertise fully on topics related to Python. I am very thankful for being his protégé. I appreciate his belief in me, for always standing behind me and pushing me to achieve more. The phrase "Journey of Thousand Miles Begins with a Single Step" is something he always reminds me of.

Thank you to my parents, Dr. Surendra Kumar Chandrakar and Smt. Bhuneshwari Chandrakar, my brother, Shri Pranav Chandrakar, my beloved wife, Mrs. Priyanka Chandrakar, my adorable son Yatharth Chandrakar, and all of my friends have inspired me and given me confidence over the years. Last but not least, I would like to express my sincere gratitude to the staff at BPB Publications for their contributions and insights that made parts of this book possible.

- *Saurabh Chandrakar*

- It was my privilege to thank Dr. S. B. Mujumdar, Chancellor of the Symbiosis International University, Pune, and Shri. Vijay Kumar Gupta, Chairman of Beekay Industries Bhilai and BIT Trust, for his encouragement and support. I would like to thank my mentors Dr. Arun Kumar Ray, Dean, School of Electronics Engineering, KIIT Deemed to be University, Bhubaneswar, and Dr. Anupam Shukla, Director, SVNIT Surat. I would like to thank Dr. Vidya Yeravdekar, Principal Director of Symbiosis Society, and the Pro-Chancellor of Symbiosis International University, Pune, Dr. Rajani R. Gupte, Vice Chancellor of the Symbiosis International University, Pune, Dr. Ketan Kotecha, Dean, Faculty of Engineering, Symbiosis International University, Pune, and Dr. Mukesh

M. Raghuwanshi, Director, SIT Nagpur, for their advice, and encouragement throughout the preparation of the book.

I would also like to thank Dr. Sanjeev Khandal, HOD, Department of Aeronautical Engineering, SGU Kolhapur, my well-wisher Dr. Prasenjeet D. Patil, Associate Professor, MIT ADT University, Pune, and my colleagues in Symbiosis Institute of Technology Nagpur for providing valuable suggestions and lots of encouragement throughout the project.

I am thankful to Prof. Dr. N. Raju, Sr. Assistant Professor, SASTRA University, Thanjavur, Tamil Nadu, for his support, assistance during writing, and for his valuable suggestions.

I would also like to thank Dr. Ravi M. Potdar, Sr. Associate Professor, BIT Durg, and Dr. Md. Khwaja Mohiddin, Associate Professor, BIT Raipur for providing valuable suggestions and lots of encouragement throughout the project. Writing a beautiful, well-balanced, and acquainted book is not a work of one or two days or a month; it takes a lot of time and patience, as well as hours of hard work. Many thanks to my family members, parents, wife, children, and well-wishers for their kind support. Without them and their faith and support, writing this classic book would have remained just a dream. I also like to thank my students, who have always been with me, for relating problems and finding solutions too. Perfection in any work does not come in a day. It needs a lot of effort, time and hard work, and sometimes, proper guidance.

It is my privilege to thank Prof. (Dr.) Ram Dhekekar, Professor, Department of Electronics & Telecommunication Engineering, SSGMCE Shegaon, and Dr. C. G. Dethe, Director UGC Staff College Nagpur. Last, but not least, I would like to offer an extra special thanks to the staff at "BPB Publications" for their insight and contribution to polishing this book.

Most significantly, I want to thank Lord Ganesha for all of the work I was able to put into the book's preparation. I would not be as zealous as I am now if it weren't for God's amazing creation of the universe.

"For since the creation of the world God's invisible qualities - his eternal power and divine nature - have been clearly seen, being understood from what has been made, so that men are without excuse."

-Dr. Niles Bhaskarao Bahadure

Preface

The purpose of this book is to introduce readers with little to no programming experience, to Python Graphical User Interface (GUI). A GUI application can be created in any programming language, say VB.Net, C#.Net etc. In this book, we shall see how to create a GUI application using Python tkinter library. We will provide the readers with the foundational knowledge and skills which is required to start writing code for creating any desktop GUI app in Python language. By mastering Python tkinter library, readers will be able to apply this technology to solve real-world problems and create various useful applications according to their needs.

The first part of the book covers basic GUI tkinter concepts followed by a touch of inbuilt variable classes for creating different tkinter GUI widgets. Then we shall see some insights of different widgets viz button, input, display, container, item and user-interactive widgets. Finally, in the later part of the book, we shall explore handling file selection and getting widget along with trace information in tkinter.

This book covers a wide range of topics, from basic definition of different widgets along with various solved examples and well explanatory code. Overall, the book provides a solid foundation for beginners to start their journey for getting trained in python GUI using tkinter library.

This book is divided into **11 chapters**. Each chapter description is listed as follows.

Chapter 1: tkinter Introduction – will cover the basic GUI example of creating a parent window along with its size maximizing by adjusting the width or height. It will also introduce about each standard attribute of python tkinter GUI, which is, dimensions, colors, fonts, cursors and so on. The concept of tkinter geometry manager with examples will be covered. Finally, we will learn how to access and set pre-defined variables sub

classed from the tkinter variable class viz StringVar, IntVar, DoubleVar and BooleanVar.

Chapter 2: Inbuilt Variable Classes for Python tkinter GUI Widgets –

will cover the concept of creating of a simple GUI windows app using classes and objects concepts.

Chapter 3: Getting Insights of Button Widgets in tkinter – is dedicated to the concept of dealing with one of the most commonly used GUI widgets viz tkinter Button widget. We will view the binding of events to the above widget with multiple examples and different methods, including lambda expressions. Next, we shall see the Checkbutton widget which will give the provision to the user to select more than one option. The user will also view different options to get the image in the above widget. Next, we will see how to use tkinter Radiobutton widget. The user will see different examples where exactly one of the predefined set of options will be chosen. Last but not the least, we will explore tkinter OptionMenu widget where the user views how a pop menu and button widget will be created for an individual option selection from a list of options.

Chapter 4: Getting Insights of Input Widgets in tkinter – is dedicated to cover the concept of creating a simple GUI app using tkinter Entry widget with very neat way of various options explanations, followed by different solved examples. Moreover, the validation concept in Entry widget is very neatly explained. Next, we shall see about scrollbar widget where user will look into the scrolling capability in vertical or horizontal direction with different widgets such as List Box, Entry and Text. Another one is tkinter Spinbox widget, where the range of input values will be fed to the user, out of which, the user can select the one. Next, we will be looking into how to implement a graphical slider to any Python application program by using tkinter Scale widget. Next is the concept of tkinter Text widget where the user can insert multiple text fields. Finally, we will be dealing with tkinter Combobox widget and its applications.

Chapter 5: Getting Insights of Display Widgets in tkinter – is dedicated to the concept of creating a simple GUI app using tkinter Label widget, which depicts the ways of displaying a text or image on a window form. We

shall also learn about the display of prompt unedited text messages to the user with tkinter Message widget. Moreover, we will look into multiple message boxes like information, warning, error and so on, in a Python application by using tkinter MessageBox widget.

Chapter 6: Getting Insights of Container Widgets in tkinter – is dedicated to the concept of tkinter Frame widget, where the user can arrange different widgets position, can provide padding, can be used as geometry manager for other widgets and so on. We shall look into the variant of Frame widget, which is tkinter LabelFrame and is a container for complex window layouts. The user will be able to see frame features along with label display. Moreover, we shall view creating tabbed widget with the help of using tkinter Notebook widget. Here, user can select different pages of contents by clicking on tabs. The importance of tkinter PanedWindow widget will be explored where multiple examples will be seen containing horizontal or vertical stack of child widgets. Finally, we will look into tkinter Toplevel widget where the concepts are being explained for the creation and display of top level windows.

Chapter 7: Getting Insights of Item Widgets in tkinter – is dedicated to tkinter Listbox widget where user can display different types of list of items and a number of items can be selected from the list. Different selectmode examples will be seen along with scrollbar attached to the above widget.

Chapter 8: Getting Insights of tkinter User Interactive Widgets – focuses on the user to create different menus such as pop-up, top level and pull -down menu with the help of tkinter Menu widget. The user can create different applications such as Notepad, Wordpad, any management software and so on. Moreover, we will explore drop-down menu widget which is associated with a Menu widget called tkinter Menubutton widget, which can display the choices when user clicks on the above Menubutton. User can add checkbutton or radiobutton with the help of above Menubutton. Finally, we will study the concepts of drawing different graphics like line, rectangle and so on, with the help of tkinter Canvas widget.

Chapter 9: Handling File Selection in tkinter – will handle file selection with different dialogs for opening a file, saving a file and so on, with the

help of multiple examples using various Python examples.

Chapter 10: Getting Widget Information and Trace in tkinter – is dedicated to getting widget information and different trace methods viz trace_add, trace_remove, trace_info and so on, using various Python examples.

Chapter 11: UserLogin Project in tkinter GUI Library with sqlite3 Database – will cover an application created using tkinter library along with interacting with sqlite3 database.

Code Bundle and Coloured Images

Please follow the link to download the ***Code Bundle*** and the ***Coloured Images*** of the book:

<https://rebrand.ly/dq4ctt8>

The code bundle for the book is also hosted on GitHub at **<https://github.com/bpbpublications/Building-Modern-GUIs-with-tkinter-and-Python>**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **<https://github.com/bpbpublications>**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at

www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Table of Contents

1. tkinter Introduction

Introduction

Structure

Objectives

Introduction to tkinter

Basic Python GUI program

Some standard attributes of Python tkinter GUI

Dimensions

borderwidth

highlightthickness

padX, padY

wraplength

height

underline

width

Colors

activebackground

background

activeforeground

foreground

disabledforeground

highlightbackground

selectbackground

selectforeground

Fonts

By creating a font object

By using tuple

Anchors

Placing widget position when anchor = N

Placing widget position when anchor = S

Placing widget position when anchor = E

Placing widget position when anchor = W

Placing widget position when anchor = NE

Placing widget position when anchor = NW

Placing widget position when anchor = SE

Placing widget position when anchor = SW

Placing widget position when anchor = CENTER

Relief styles

Bitmaps

Cursors

Python tkinter geometry management

pack()

grid()

place()

Geometry method in tkinter

Conclusion

Points to remember

Questions

2. Inbuilt Variable Classes for Python tkinter GUI Widgets

Introduction

Structure

Objectives

Inbuilt variable classes
StringVar()
BooleanVar()
IntVar()
DoubleVar()
GUI creation using classes and objects
Conclusion
Points to remember
Questions

3. Getting Insights of Button Widgets in tkinter

Introduction
Structure
Objectives
tkinter Button Widget
Events and bindings
event type
tkinter Checkbutton widget
tkinter Radiobutton widget
tkinter OptionMenu widget
Conclusion
Points of remember
Questions

4. Getting Insights of Input Widgets in tkinter

Introduction
Structure
Objectives

[tkinter Entry widget](#)

Validation in the Entry widget

[tkinter Scrollbar widget](#)

Scrollbar attached to Listbox

Scrollbar attached to Text

Scrollbar attached to Canvas

Scrollbar attached to Entry

[tkinter Spinbox widget](#)

[tkinter Scale widget](#)

[tkinter Text widget](#)

[tkinter Combobox Widget](#)

[Conclusion](#)

[Points to remember](#)

[Questions](#)

5. Getting Insights of Display Widgets in tkinter

[Introduction](#)

[Structure](#)

[Objectives](#)

[tkinter Label Widget](#)

[tkinter Message Widget](#)

[tkinter MessageBox Widget](#)

showinfo()

showwarning()

showerror()

askquestion()

askokcancel()

askyesno()

askretrycancel()

Conclusion

Points of remember

Questions

6. Getting Insights of Container Widgets in tkinter

Introduction

Structure

Objectives

tkinter Frame Widget

tkinter LabelFrame Widget

tkinter Tabbed/Notebook Widget

tkinter PanedWindow widget

tkinter Toplevel widget

Conclusion

Points of remember

Questions

7. Getting Insights of Item Widgets in tkinter

Introduction

Structure

Objectives

tkinter Listbox widget

Conclusion

Points of remember

Questions

8. Getting Insights of tkinter User Interactive Widgets

Introduction

Structure

Objectives

tkinter Menu widget

tkinter Menubutton widget

tkinter Canvas widget

Conclusion

Points to remember

Questions

9. Handling File Selection in tkinter

Introduction

Structure

Objectives

Handling file selection in tkinter

Conclusion

Points of remember

Questions

10. Getting Widget Information and Trace in tkinter

Introduction

Structure

Objectives

Getting widget information

Trace in tkinter

trace_add()

trace_remove()

trace_info()

Conclusion

Points to remember

Questions

11. UserLogin Project in tkinter GUI Library with sqlite3 Database

Introduction

Structure

Objectives

GUI interaction with sqlite3 database

Displaying a GUI application

Conclusion

Points to remember

Questions

Index

CHAPTER 1

tkinter Introduction

Introduction

We have learned in our previous book *Python for Everyone*, about the concepts in Python which are procedure oriented or object oriented. However, these concepts will be used as indispensable salt for our next learning, which is related to **Graphical User Interface (GUI)**. We are surrounded by GUI apps in day-to-day life. Whenever we are using our mobile phone/Desktop applications and accessing any app or software, the first thing which we look forward to is how to access these apps or software. Any app on mobile phone or a computer system consists of hardware and is controlled by an operating system. The high-level languages will be sitting on top of the operating system and Python is no exception. So, in this chapter, we will learn about tkinter library in Python.

Structure

In this chapter, we will discuss the following topics:

- Introduction to tkinter
- Basic Python GUI Program
- Some standard attributes of Python tkinter GUI
- Colors
- Fonts
- Anchors

- Relief Styles
- Bitmaps
- Cursors
- Python tkinter Geometry Manager

Objectives

By the end of this chapter, the reader will learn about creating basic GUI Python program using the tkinter library. In addition to that, we will also explore some standard attributes of Python tkinter GUI such as dimensions, colors or fonts with various options with examples. It is important to know how to position the text with a list of constants, relative to the reference point using anchor attribute. Moreover, we shall see how with the usage of relief attribute 3-D, simulated effects around the outside of the widget can be provided. We will also learn about bitmap and cursor attribute with examples. Finally, at the end of this chapter, we will learn accessing tkinter widgets using inbuilt layout geometry managers viz pack, grid and place.

Introduction to tkinter

Whenever we write any program in Python to control the hardware, Python will show the output with the help of operating system. However, if we desire to make an executable with the help of GUI, then just having the need of hardware and operating system is insufficient. Python requires some services which come from a number of resources and one such resource which is of interest to many Python programmers is Tcl/Tk. **Tcl** stands for **Tool Command Language** and it is a scripting language with its own interpreter. On the other hand, Tk is a toolkit for building GUIs. An important point to note is that Tcl/Tk is not Python and we cannot control and access the services of Tcl/Tk using Python. So, another package is introduced and is referred to as *tkinter*, and it is an mediator between Python and Tcl/Tk. tkinter will allow us to use the services of Tcl/Tk using the syntax of Python. As Python code developers, we will not be directly concerned with the Tcl/Tk. Binding of Python to the Tk GUI toolkit will be done by tkinter. tkinter will make everything appear as an object. We can

create GUI, knowing that we can regard the window as an object, a label place on window as an object and so on. Applications can be built from a view point of an object-oriented programming paradigm. All we need to make sure is that we write our code in such a way that it allows us access tkinter, as shown in the following *Figure 1.1*:

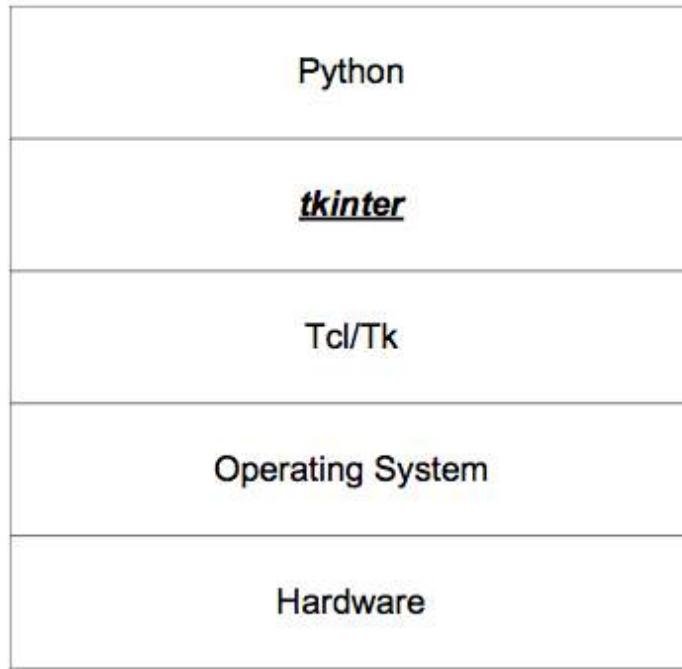


Figure 1.1: tkinter access hierarchy

So, GUI applications can be easily and quickly created when Python is combined with tkinter. Although Python offers multiple options for developing GUI such as PyQT, Kivy, Jython, WxPython, and pyGUI; tkinter is the most commonly used. In this book, we will focus only on the tkinter usage of GUI creation. Just like we import any other module, tkinter can be imported in the same way in Python code:

```
import tkinter
```

This will import the **tkinter** module.

The module name in Python 3.x is **tkinter**, whereas in Python 2.x, it is Tkinter.

More often, we can use:

```
from tkinter import *
```

Here, the '*' symbol means everything, as Python now can build Graphical User Interfaces by treating all of the widgets like Buttons, Labels, Menus and so on, as if they were objects. It is like importing `tkinter` submodule.

However, there are some important methods which the user needs to know while creating Python GUI application, and they are as follows:

- **Tk(screenName=None, baseName=None, className='Tk', useTk=1)**

`tkinter` offers this method in order to create a main window. For any application, a main window code can be created by using:

```
import tkinter  
myroot = tkinter.Tk()
```

Here, `Tk` class is instantiated without any arguments. `myroot` is the main window object name. This method will allow blank parent window creation with close, maximize and minimize buttons on the top.

- **mainloop()**

`tkinter` offers this method when our application is ready to run. This method is an infinite loop used to run the application. It will wait for an event to occur and as long as the window is not closed, the event is processed.

Basic Python GUI program

Let us see a basic Python program which will create a window:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class

# we should first know how to create a window if want to per-
form graphics coding.

# but output window will not be displayed right now

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.2*:

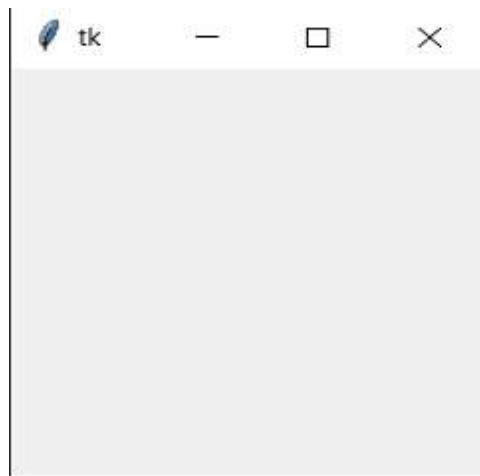
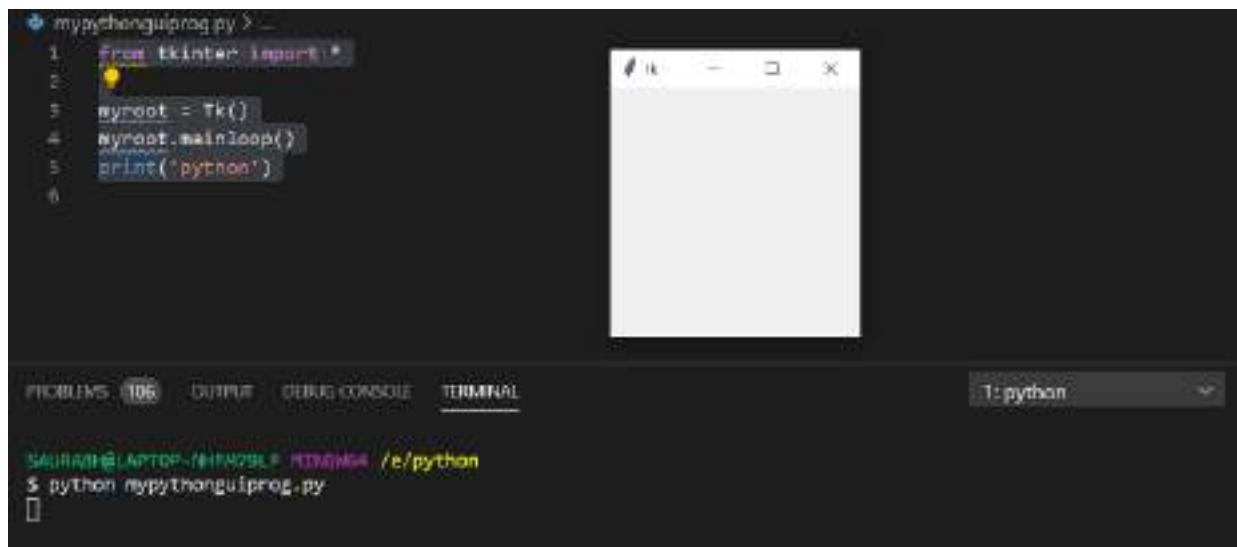


Figure 1.2: Output of Chap1_Example1.py

Note: The preceding code is covered in Program Name: Chap1_Example1.py

In the preceding code, we have imported tkinter submodule and created a parent widget which usually will be the main window of an application. A blank parent window is created with close, maximize and minimize buttons on the top. An infinite loop will be called to run the application, as long as the window is not closed.

The moment the window is closed, the statements after `myroot.mainloop()` will be executed, as shown in the following *Figure 1.3*:



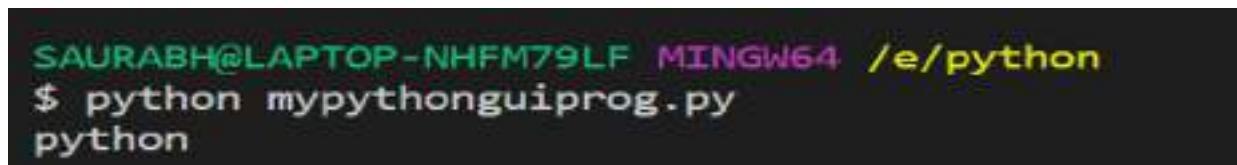
The screenshot shows a code editor with a Python script named `mypythonguiprog.py`. The code creates a Tkinter window and prints "python". Below the editor is a terminal window titled "1:python" showing the command `$ python mypythonguiprog.py` and its output "python".

```
mypythonguiprog.py>1 from tkinter import *  
2  
3 myroot = Tk()  
4 myroot.mainloop()  
5 print("python")  
6
```

```
PROBLEMS (0) OUTPUT DEBUG CONSOLE TERMINAL 1:python  
SAURABH@LAPTOP-NHFM79LF MINGW64 /e/python  
$ python mypythonguiprog.py
```

Figure 1.3: Output on running the code

The moment the window is closed by clicking on the 'X' mark, we will get the output as shown in *Figure 1.4*:



The terminal shows the command `$ python mypythonguiprog.py` being run, and the output "python" is displayed.

```
SAURABH@LAPTOP-NHFM79LF MINGW64 /e/python  
$ python mypythonguiprog.py  
python
```

Figure 1.4: Execution of print statement after clicking 'X' mark

In the preceding example, we can maximize the window in both horizontal and vertical directions by using both height and width attributes. However, we can restrict the expansion of window in any direction.

Suppose we want to maximize the width only. Then, the code is as follows:

```
from tkinter import *  
  
myroot = Tk()  
myroot.resizable(width = True, height = False) # width can be maximized  
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.5*:

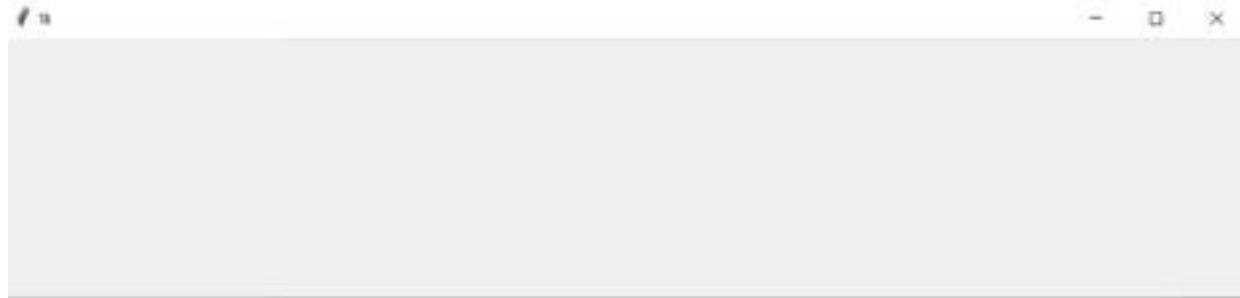


Figure 1.5: Output of Chap1_Example2.py

Note: The preceding code is covered in Program Name: Chap1_Example2.py

Suppose, we want to maximize the height only. Then the code is as follows:

```
from tkinter import *

myroot = Tk()

myroot.
resizable(width = False, height = True) # height can be maximized

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.6*:

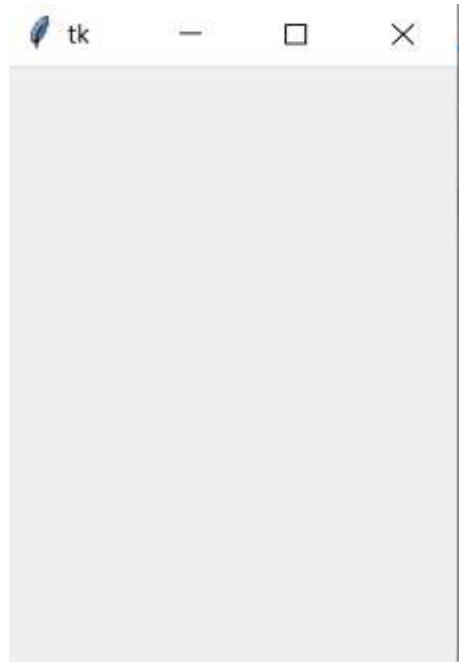


Figure 1.6: Output of Chap1_Example3.py

Note: The preceding code is covered in Program Name: **Chap1_Example3.py**

Now, there is a need to maximize neither height nor width. Then the following code will be used:

```
from tkinter import *

myroot = Tk()
myroot.resizable(width = False, height = False) # nei-
ther width nor height can be maximized
myroot.mainloop()
```

Output:

The output can be seen in [*Figure 1.7:*](#)

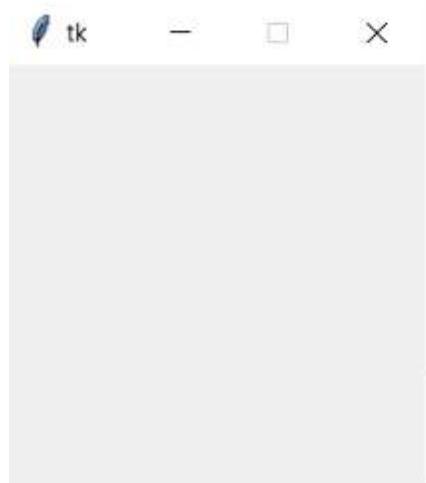


Figure 1.7: Output of Chap1_Example4.py

Note: The preceding code is covered in Program Name: Chap1_Example4.py

An important point to observe is that when we are maximizing the window in either of the directions, then the maximize button was enabled. Whereas in this case, the maximize button is disabled.

Some standard attributes of Python tkinter GUI

Now, we shall see how some of the standard attributes such as sizes, colors or fonts are specified. Just observe the standard attributes as mentioned. We shall see their usage when we will be dealing with widgets.

Dimensions

Whenever we set a dimension to an integer, it is assumed to be in pixels. A length is expressed as an integer number of pixels by tkinter. The list of common options are discussed as follows.

borderwidth

This option will give a 3-D look to the widget. It can also be represented as **bd:**

```
from tkinter import *
myroot = Tk()
myl1 = Label(myroot, text = 'Label1',bd = 8,relief = 'groove')
myl1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.8*:



Figure 1.8: Output of Chap1_Example5.py

Note: The preceding code is covered in Program Name: Chap1_Example5.py

highlightthickness

This option represents the width of the highlighted rectangle when the widget has focus. Refer to the following code:

```
from tkinter import *

myroot = Tk()

myb1 = Button(myroot, text = 'Without highlight thickness')
myb1.grid(row = 0, column = 1)

myb2 = Button(myroot, text = 'With highlight thickness',
              highlightthickness=10,
              )
myb2.grid(row = 1, column = 1, padx = 10, pady = 10)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.9*:

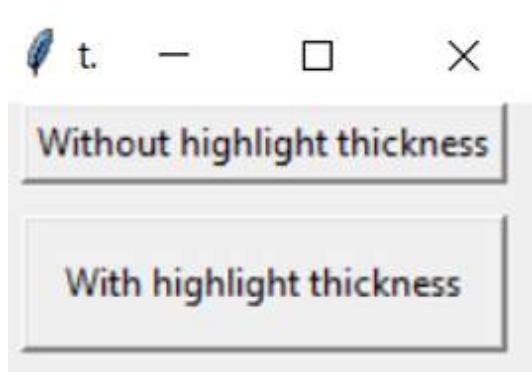


Figure 1.9: Output of Chap1_Example6.py

Note: The preceding code is covered in Program Name: **Chap1_Example6.py**

padX, padY

This option will provide extra space that the widget requests from its layout manager, beyond the minimum, for the display of widget contents in x and y

directions. We can see in the previous example, that we have used padding in x and y direction for better look and display.

wraplength

This option will provide maximum length of line for widgets which will be performing word wrapping. Refer to the following code:

```
from tkinter import *
myroot = Tk()
myl1 = Label(myroot,  text = 'Python',wraplength = 2)
myl1.pack()
myl2 = Label(myroot,  text = 'awesome',wraplength = 0)
myl2.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.10*:

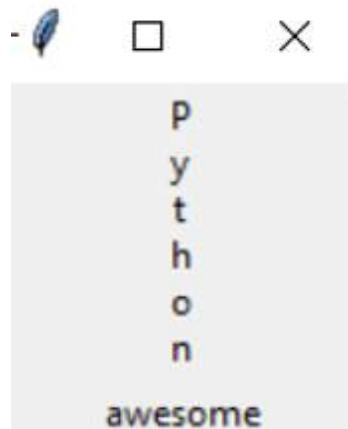


Figure 1.10: Output of Chap1_Example7.py

Note: The preceding code is covered in Program Name: **Chap1_Example7.py**

height

This option will set the desired height of the widget as per need. It must be greater than 1.

underline

This option represents character index to underline in the widget's text. The 1st character will be 0, the 2nd character will be, 1 and so on.

width

This option will set the desired width of the widget as per need, as shown:

```
from tkinter import *
myroot = Tk()
myl1 = Label(myroot, text = 'Python',width = 20, height = 2, underline = 2, font = ('Calibri',15))
myl1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.11*:

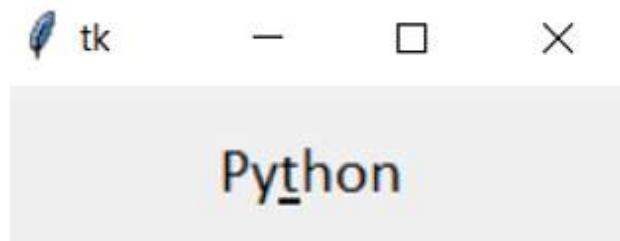


Figure 1.11: Output of Chap1_Example8.py

Note: The preceding code is covered in Program Name: **Chap1_Example8.py**

In the above code, we have set width as 20, height as 2 and underline the 3rd character which is 't'.

Colors

Colors can be represented in tkinter using hexadecimal digits or by standard name. For example, #ff0000 is for Red color or it can be represented by using color = ‘Red’. The different options available for color is are discussed as follows.

activebackground

This option will set the widget background color when it is active.

background

This option will set the widget background color and can also be represented as **bg**, as follows:

```
from tkinter import *
myroot = Tk()
myb1 = Button(myroot, activebackground = "#ff0000", bg = '#00ff00', text = 'python')
myb1.pack()
myroot.mainloop()
```

Output Initially:

The output can be seen in *Figure 1.12*:

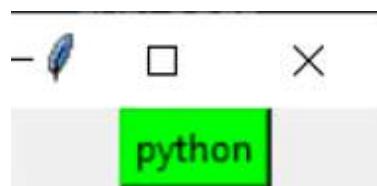


Figure 1.12: Initial Output

Output when button is clicked:

The output can be seen in *Figure 1.13*:

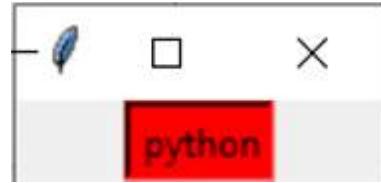


Figure 1.13: Output of Chap1_Example9.py

Note: The preceding code is covered in Program Name: Chap1_Example9.py

On running the preceding code, initially the background color of button is Green and when it is clicked, that is, when the button is active, the background color is changed to Red.

activeforeground

This option will set the widget foreground color when it is active.

foreground

This option will set the widget foreground color and can also be represented as **fg**.

```
from tkinter import *
myroot = Tk()
myb1 = Button(myroot, activefore-
ground = "#ff0000", fg = '#0000ff', text = 'python')
myb1.pack()
myroot.mainloop()
```

Output initially:

The output can be seen in *Figure 1.14*:



Figure 1.14: Initial output

Output when button is clicked:

The output can be seen in *Figure 1.15*:

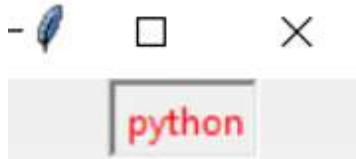


Figure 1.15: Output of Chap1_Example10.py

Note: The preceding code is covered in Program Name: Chap1_Example10.py

On running the preceding code, initially the foreground color of button is Blue and when it is clicked, that is, when the button is active, the foreground color is changed to Red.

disabledforeground

This option will set the widget foreground color when it is disabled, as shown:

```
from tkinter import *
myroot = Tk()
myb1 = Button(myroot, state = 'disabled', text = 'python', disabledforeground = 'Magenta')
myb1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.16*:



Figure 1.16: Output of Chap1_Example11.py

Note: The preceding code is covered in Program Name: Chap1_Example11.py

In the preceding code, the button is disabled and the button's foreground color when it is disabled is Magenta.

highlightbackground

When the widget does not have any focus, this option will focus on the highlight color.

highlightcolor

When the widget has focus, this option will set the foreground color of the highlighted region, as shown:

```
from tkinter import *
myroot = Tk()
mye1 = Entry(myroot, bg='LightGreen', highlightthickness=10, highlightbackground="red", highlightcolor = 'Yellow')
mye1.pack(padx=5, pady=5)
mye2 = Entry(myroot)
mye2.pack()
mye2.focus()
myroot.mainloop()
```

Output when there is no focus on Entry:

The output can be seen in *Figure 1.17*:



Figure 1.17: Initial output

Output when there is focus on Entry:

The output can be seen in *Figure 1.18*:



Figure 1.18: Output of Chap1_Example12.py

Note: The preceding code is covered in Program Name: Chap1_Example12.py

In the preceding example, when there is no focus in Entry, the focus highlight's color is Red and when there is a focus on Entry, then the color in the focus highlight is Yellow.

selectbackground

This option will set the background color for the selected items of the widget.

selectforeground

This option will set the foreground color for the selected items of the widget, as shown:

```

from tkinter import *

myroot = Tk()
str1 = StringVar()
mye1 = Entry(myroot,selectbackground= 'Green',selectforeground= 'Red', textvariable = str1)
mye1.pack()
str1.set('python')

myroot.mainloop()

```

Output:

The output can be seen in *Figure 1.19*:

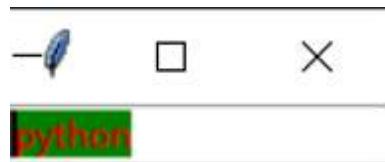


Figure 1.19: Output of Chap1_Example13.py

Note: The preceding code is covered in Program Name: **Chap1_Example13.py**

From the preceding code, we can observe that when the text ‘python’ is selected, it is highlighted in Green color in the background and Red color in the foreground.

Fonts

We can access the font in tkinter by creating a font object or by using a tuple.

By creating a font object

Suppose we have a text. We can underline the text, change the size, give some font family, overstrike it, and so on.

To create a font object, first, we need to import a **tkinter.font** module and use the **Font** class constructor:

```
from tkinter.font import Font  
myobj_font = Font(options,...)
```

The options are:

- **family**: This option represents the name of the font family as a string.
- **size**: This option represents an integer in points for font height.
- **weight**: This option represents ‘bold’ and ‘normal’.
- **slant**: This option represents ‘italic’ and ‘unslanted’.
- **underline**: This option represents whether the text is to be underlined or not. 1 represents for underline text and 0 for normal.
- **overstrike**: This option represents whether the text is overstruck or not. 1 represents overstruck text and 0 for normal.

Refer to the following code:

```
from tkinter import * # importing module  
from tkinter.font import Font  
myroot = Tk() # window creation and initialize the interpreter  
  
myfont1 = Font(family = 'Cal-  
ibri', size=12, weight = 'bold', slant='italic', underline = 1, over-  
strike = 1) # 1 means we require underline  
myl1 = Label(myroot, text = 'Python', font = myfont1)  
myl1.pack() # for displaying the label  
  
myroot.mainloop() # display window until we press the close button
```

Output:

The output can be seen in *Figure 1.20*:



Figure 1.20: Output of Chap1_Example14.py

Note: The preceding code is covered in Program Name: Chap1_Example14.py

We can also view the font family which we can access based on our requirement, as follows:

```
from tkinter import * # importing module
from tkinter import font
myroot = Tk() # window creation and initialize the interpreter

myfont_list = list(font.families())
for loop in myfont_list:
    print(loop,end = ',')
    
myroot.mainloop() # display window until we press the close button
```

Output:

The output is as follows:

```
System,8514oem,Fixedsys,Terminal,Modern,Roman,Script,Courier,MS
Serif,MS Sans Serif,Small Fonts,TeamViewer15,Marlett,Arial,Arabic
Transparent,Arial Baltic,Arial CE,Arial CYR,Arial
Greek,Arial TUR,Arial Black,Bahnschrift Light,Bahnschrift
Semilight,Bahnschrift,Bahnschrift SemiBold,Bahnschrift Light
SemiCondensed,Bahnschrift Semilight SemiConde,Bahnschrift
SemiCondensed,Bahnschrift SemiBold SemiConden,Bahnschrift
Light Condensed, Bahnschrift Semilight Condensed,Bahnschrift
Condensed,Bahnschrift SemiBold Condensed,Calibri,Calibri
Light,Cambria,Cambria Math,Candara,Candara Light, Comic Sans
MS,Consolas,Constantia,Corbel,Corbel Light,Courier New,Courier New
Baltic,Courier New CE,Courier New CYR,Courier New Greek,Courier
New TUR, Ebrima, Franklin Gothic Medium, Gabriola, Gadugi,
Georgia,Impact,Ink Free, Javanese Text,Leelawadee UI,Leelawadee UI
Semilight,Lucida Console,Lucida Sans Unicode,Malgung Gothic,@Malgung
```

Gothic,Malgun Gothic Semilight,@Malgun Gothic Semilight,Microsoft Himalaya,Microsoft JhengHei,@Microsoft JhengHei,Microsoft JhengHei UI,@Microsoft JhengHei UI,Microsoft JhengHei Light,@Microsoft JhengHei Light,Microsoft JhengHei UI Light,@Microsoft JhengHei UI Light,Microsoft New Tai Lue,Microsoft PhagsPa,Microsoft Sans Serif,Microsoft Tai Le,Microsoft YaHei,@Microsoft YaHei,Microsoft YaHei UI,@Microsoft YaHei UI,Microsoft YaHei Light,@Microsoft YaHei Light,Microsoft YaHei UI Light,@Microsoft YaHei UI Light,Microsoft Yi Baiti,MingLiU-ExtB,@MingLiU-ExtB,PMingLiU-ExtB,@PMingLiU-ExtB,MingLiU_HKSCS-ExtB,@MingLiU_HKSCS-ExtB,Mongolian Baiti,MS Gothic,@MS Gothic,MS UI Gothic,@MS UI Gothic,MS PGothic,@MS PGothic,MV Boli,Myanmar Text,Nirmala UI,Nirmala UI Semilight,Palatino Linotype,Segoe MDL2 Assets,Segoe Print,Segoe Script,Segoe UI,Segoe UI Black,Segoe UI Emoji,Segoe UI Historic,Segoe UI Light,Segoe UI Semibold,Segoe UI Semilight,Segoe UI Symbol,SimSun,@SimSun,NSimSun,@NSimSun,SimSun-ExtB,@SimSun-ExtB,Sitka Small,Sitka Text,Sitka Subheading,Sitka Heading,Sitka Display,Sitka Banner,Sylfaen,Symbol,Tahoma,Times New Roman,Times New Roman Baltic,Times New Roman CE,Times New Roman CYR,Times New Roman Greek,Times New Roman TUR,Trebuchet MS,Verdana,Webdings,Wingdings,Yu Gothic,@Yu Gothic,Yu Gothic UI,@Yu Gothic UI,Yu Gothic UI Semibold,@Yu Gothic UI Semibold,Yu Gothic Light,@Yu Gothic Light,Yu Gothic UI Light,@Yu Gothic UI Light,Yu Gothic Medium,@Yu Gothic Medium,Yu Gothic UI Semilight,@Yu Gothic UI Semilight,HoloLens MDL2 Assets,HP Simplified,HP Simplified Light,MT Extra,Century,Wingdings 2,Wingdings 3,Arial Unicode MS,@Arial Unicode MS,Nina,Segoe Condensed,Buxton Sketch,Segoe Marker,SketchFlow Print,DengXian,@DengXian,Microsoft MHei,@Microsoft MHei,Microsoft NeoGothic,@Microsoft NeoGothic,Segoe WP Black,Segoe WP Semibold,Segoe WP Light,Segoe WP SemiLight, DigifaceWide, AcadEref, AIGDT, AmdtSymbols, GENISO,AMGDT,BankGothic Lt

BT,BankGothic Md BT,CityBlueprint,CommercialPi BT,CommercialScript BT,CountryBlueprint,Dutch801 Rm BT,Dutch801 XBd BT,EuroRoman,ISOCEUR,ISOCTEUR,Monospac821 BT,PanRoman,Romantic,RomanS,SansSerif,Stylus BT,SuperFrench,Swis721 BT,Swis721 BdOul BT,Swis721 Cn BT,Swis721 BdCnOul BT,Swis721 BlkCn BT,Swis721 LtCn BT,Swis721 Ex BT,Swis721 BlkEx BT,Swis721 LtEx BT,Swis721 Blk BT,Swis721 BlkOul BT,Swis721 Lt BT,TechnicBold,TechnicLite,Technic,UniversalMath1 BT,Vineta

CP,ISOCT2,ISOCT3,ISOCT,ItalicC,ItalicT,Italic,Monotxt,Proxy 1,Proxy 2,Proxy 3,Proxy 4,Proxy 5,Proxy 6,Proxy 7,Proxy 8,Proxy 9,RomanC,RomanD,RomanT,ScriptC,ScriptS,Simplex,Syastro,Syast,Sympa,Symath,Symeteo,Symusic,Txt,

Note: The preceding code is covered in Program Name: Chap1_Example15.py

By using tuple

The first element in the tuple is a font family, and the 2nd element is a size in points which is optionally followed by a string having one or more of the style modifiers such as bold, italic, overstrike, and underline. Refer to the following code:

```
from tkinter import * # importing module
from tkinter.font import Font
myroot = Tk() # window creation and initialize the interpreter

myl1 = Label(myroot, text = 'Python', font = ("Calibri", "18", "normal italic overstrike underline"))
myl1.pack() # for displaying the label

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.21*:

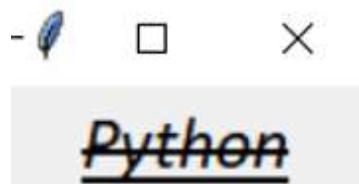


Figure 1.21: Output of Chap1_Example16.py

Note: The preceding code is covered in Program Name: Chap1_Example16.py

Anchors

If there is a requirement to position the text relative to the reference point, then we should go for anchors.

The possible list of constants used for anchor attributes are:

N, S, E, W, NE, NW, SE, SW, CENTER

Here, N stands for North, S for South, E for East and W for West.

Refer to *Figure 1.22*:

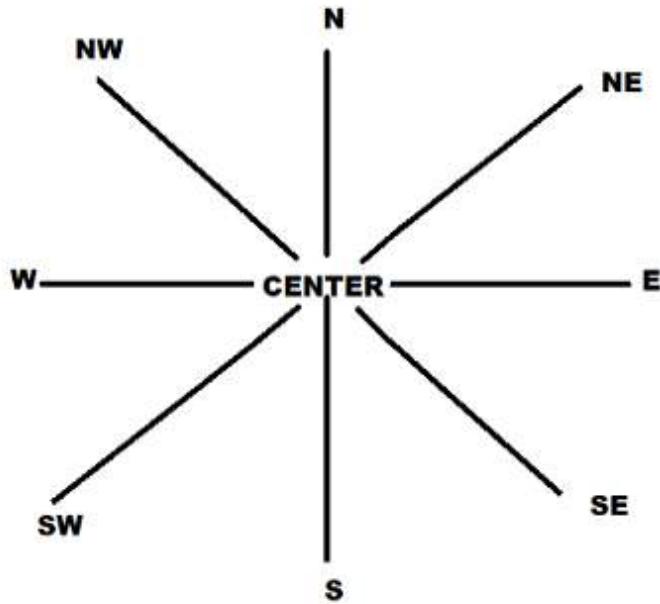


Figure 1.22: Anchor Constants

Whenever we are creating a small widget inside a large frame and use anchor = SW option, then the widget will be placed in the bottom left corner of the frame. If we use anchor = S, then the widget would be centered along the bottom edge.

Placing widget position when anchor = N

Refer to the following code:

```
from tkinter import *

myroot = Tk()
myroot.geometry('200x200')
myl1 = Label(myroot, text = 'Python', anchor = N, font = ("Calibri", "18", "bold italic underline"),
             bd = 1, relief = 'sunken', width = 10,height = 5 )
myl1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.23*:



Figure 1.23: Output of Chap1_Example17.py when anchor position is N

Note: The preceding code is covered in Program Name: Chap1_Example17.py

Note: From Figure 1.24 to Figure 1.31, the code will be the same except the option at anchor position will be changed.

Placing widget position when anchor = S

Refer to *Figure 1.24*:



Figure 1.24: Output when anchor position is S

Placing widget position when anchor = E

Refer to *Figure 1.25*:



Figure 1.25: Output when anchor position is E

Placing widget position when anchor = W

Refer to *Figure 1.26*:



Figure 1.26: Output when anchor position is W

Placing widget position when anchor = NE

Refer to *Figure 1.27*:



Figure 1.27: Output when anchor position is NE

Placing widget position when anchor = NW

Refer to *Figure 1.28*:



Figure 1.28: Output when anchor position is NW

Placing widget position when anchor = SE

Refer to *Figure 1.29*:



Figure 1.29: Output when anchor position is SE

Placing widget position when anchor = SW

Refer to *Figure 1.30*:



Figure 1.30: Output when anchor position is SW

Placing widget position when anchor = CENTER

Refer to *Figure 1.31*:



Figure 1.31: Output when anchor position is CENTER

Relief styles

Whenever we desire to have 3-D simulated effects around the outside of the widget, then we will go for the relief style of a widget. Relief attributes can have a possible list of constants such as flat, raised, groove, sunken, and ridge. Refer to the following code:

```
from tkinter import *

myroot = Tk()
myroot.geometry('200x200')
myb1 = Button(myroot, text = 'PYTHON', font = ('Calibri',12), relief = FLAT, bd = 4)
myb1.pack()
myb2 = Button(myroot, text = 'PYTHON', font = ('Calibri',12), relief = RAISED, bd = 4)
myb2.pack()
myb3 = Button(myroot, text = 'PYTHON', font = ('Calibri',12), relief = 'groove', bd = 4)
myb3.pack()
myb4 = Button(myroot, text = 'PYTHON', font = ('Calibri',12), relief = 'sunken', bd = 4)
myb4.pack()
myb5 = Button(myroot, text = 'PYTHON', font = ('Calibri',12), relief = RIDGE, bd = 4)
myb5.pack()
myroot.mainloop()
```

Output:

Refer to *Figure 1.32*:

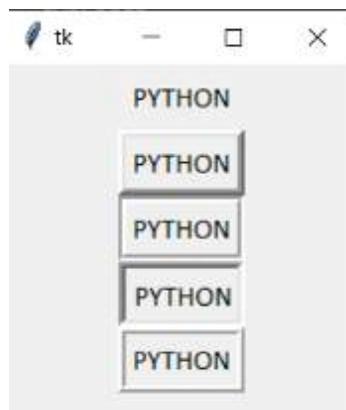


Figure 1.32: Output of Chap1_Example18.py

Note: The preceding code is covered in Program Name: **Chap1_Example18.py**

Bitmaps

This attribute is used to display a bitmap and the available bitmaps are ‘error’, ‘gray12’, ‘gray25’, ‘gray50’, ‘gray75’, ‘info’, ‘hourglass’, ‘warning’, ‘question’, and ‘questhead’. Refer to the following code:

```
from tkinter import *

myroot = Tk()
myroot.geometry('200x300')
myb1 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'error',bd = 2)
myb1.pack()
myb2 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'gray12',bd = 2)
myb2.pack()
myb3 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'gray25',bd = 2)
myb3.pack()
myb4 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'gray50',bd = 2)
myb4.pack()
myb5 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'gray75',bd = 2)
myb5.pack()
myb6 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'info',bd = 2)
myb6.pack()
myb7 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'hourglass',bd = 2)
myb7.pack()
myb8 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'warning',bd = 2)
myb8.pack()
myb9 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'question',bd = 2)
myb9.pack()
myb10 = Button(myroot, text = 'PYTHON', relief = 'sunken', bitmap = 'questhead',bd = 2)
myb10.pack()
myroot.mainloop()
```

Output:

Refer to *Figure 1.33*:

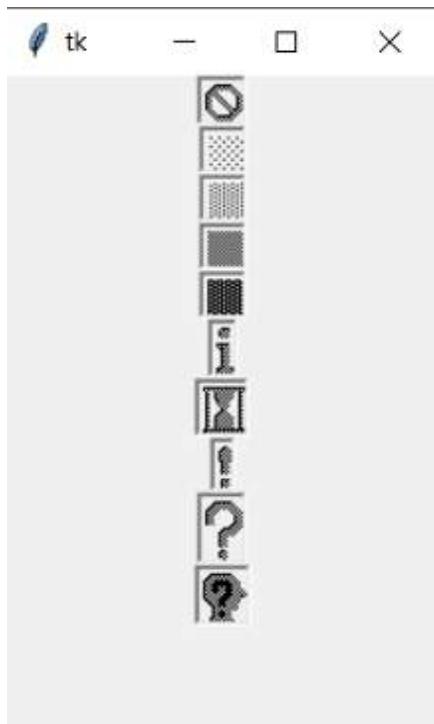


Figure 1.33: Output of Chap1_Example19.py

Note: The preceding code is covered in Program Name: **Chap1_Example19.py**

Cursors

Whenever there is a requirement to show different mouse cursors based on the need whose exact graphic may vary depending on the operating system, then there is an option of cursor attribute, out of which, some useful ones are ‘arrow’, ‘clock’, ‘cross’, ‘circle’, ‘heart’, ‘mouse’, ‘plus’, ‘star’, ‘spider’, ‘sizing’, ‘shuttle’, ‘target’, ‘tcross’, ‘trek’, ‘watch’ and so on. Refer to the following code:

```
from tkinter import *

myroot = Tk()
myroot.geometry('200x300')
myb4 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'circle',bd = 2)
myb4.pack()
myb5 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'clock',bd = 2)
myb5.pack()
myb6 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'cross',bd = 2)
myb6.pack()
myb7 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'plus',bd = 2)
myb7.pack()
myb8 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'tcross',bd = 2)
myb8.pack()
myb9 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'star',bd = 2)
myb9.pack()
myb10 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'spider',bd = 2)
myb10.pack()
myb11 = Button(myroot, text = 'PYTHON', relief = 'raised', cursor = 'watch',bd = 2)
myb11.pack()
myroot.mainloop()
```

Output:

Refer to *Figure 1.34*:

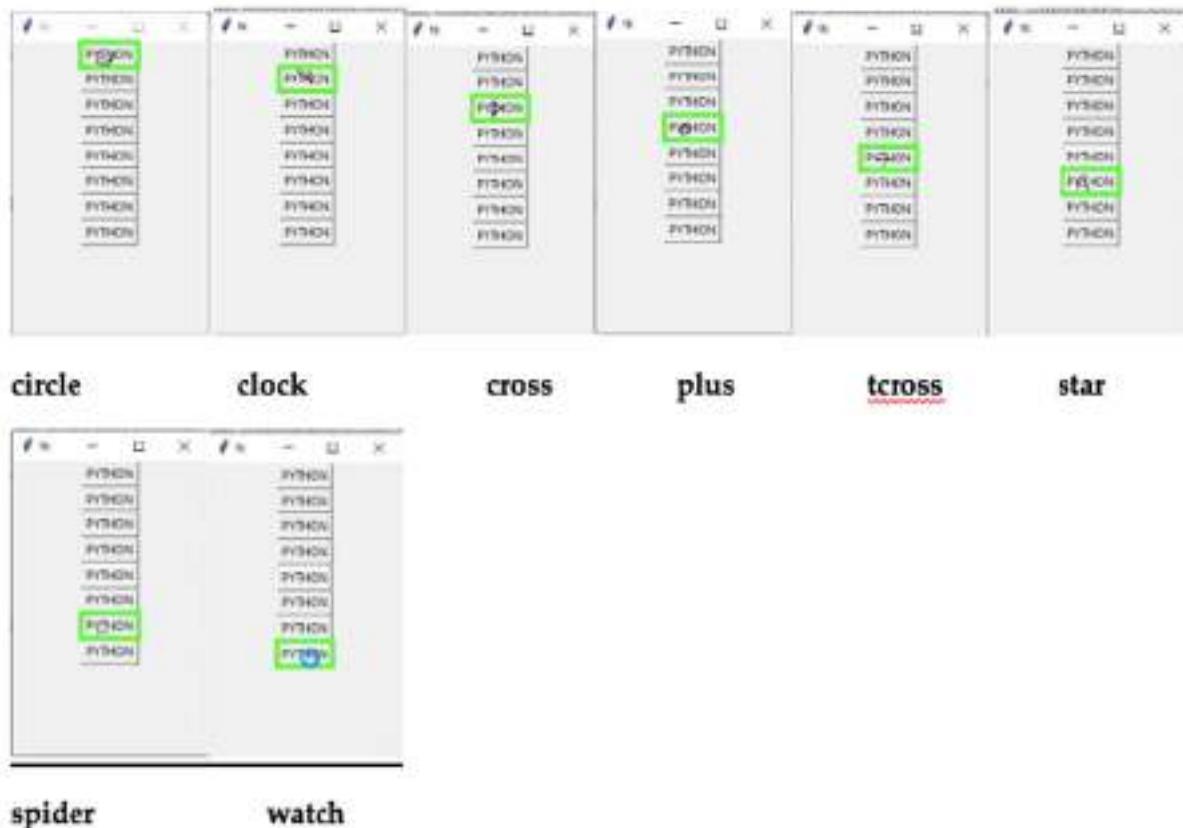


Figure 1.34: Output of Chap1_Example20.py

Note: The preceding code is covered in Program Name: Chap1_Example20.py

Python tkinter geometry management

All widgets in the tkinter can access geometry management methods. To organize the widgets in the parent windows, the geometry configuration of the widgets is to be accessed, which is offered by tkinter. It will help in managing the display of widgets on the screen. The geometry manager classes are discussed as follows.

pack()

It is one of the easiest to use, compared to the other 2 geometry managers. Before placing the widgets in the parent widget, the geometry manager will organize the widgets in blocks. Whenever there is a simple application requirement, such as arranging the number of widgets on top of each other or

placing them side by side, then we can go for the preceding geometry manager. As compared to the other two geometry managers, it comes with limited options.

The syntax is

```
widget.pack(options,...)
```

The options are:

- **fill**: This option will determine whether the widgets can increase or grow in size or not. By default, it is NONE. If we want to fill vertically, then it is Y. If horizontally, then it is X. If required both horizontally or vertically, then BOTH.
- **expand**: This option will expand the widget to fill any space when set to true or 1. When the window is resized, the widget will expand.
- **side**: This option will decide the widget alignment, that is, against which side of the parent widget packs. By default, it is TOP. The others are BOTTOM, LEFT, or RIGHT.

The other options are anchor, internal (ipadx, ipady) or external padding (padx, pady) which all have defaulted to 0. Refer to the following code:

```
from tkinter import *
myroot = Tk()
myroot.geometry('300x300')
myb1 = Button(myroot, text = 'P', fg = 'Red', bg = 'LightGreen')
myb1.pack(fill = NONE)
myb2 = Button(myroot, text = 'Y', fg = 'Red', bg = 'LightGreen')
```

```

myb2.pack(fill = X, padx = 10, pady = 10)
myb3 = Button(myroot, text = 'T', fg = 'Red', bg = 'LightGreen')
myb3.pack(side = LEFT, fill = Y, padx = 10, pady = 10)
myb3 = Button(myroot, text = 'H', fg = 'Red', bg = 'LightGreen')
myb3.pack(side = TOP, fill = X, padx = 10, pady = 10)
myb4 = Button(myroot, text = 'O', fg = 'Red', bg = 'LightGreen')
myb4.pack(side = BOTTOM, fill = X, padx = 10, pady = 10)
myb5 = Button(myroot, text = 'N', fg = 'Red', bg = 'LightGreen')
myb5.pack(side = RIGHT, fill = BOTH, expand = 1, padx = 10, pady = 10)

myroot.mainloop()

```

Output:

The output can be seen in *Figure 1.35*:

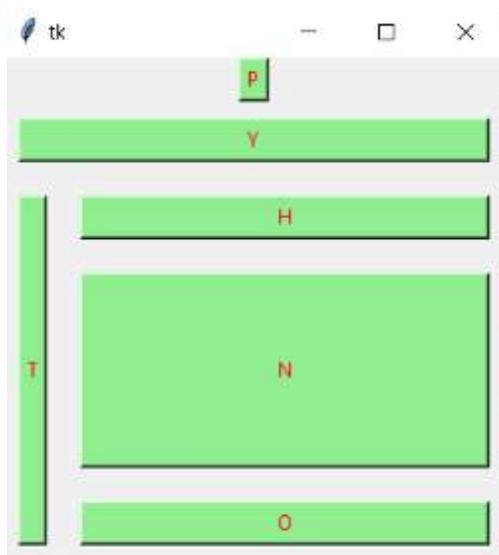


Figure 1.35: Output of Chap1_Example21.py

Note: The preceding code is covered in Program Name: **Chap1_Example21.py**

grid()

This geometry manager will organize the widgets into a 2-Dimensional table, into a number of rows and columns in the parent widget. An

intersection of imaginary rows and columns is a cell where each cell in the table can hold a widget.

The syntax is

```
widget.grid(options,...)
```

The options are as follows:

- **column:** This option will use a cell which will be identified with a given column, whose default value is the leftmost column with a numeric value of 0.
- **columnspan:** This option will indicate the number of columns the widget occupies; whose default value is 1.
- **padx:** This option will add padding to the widget horizontally outside the border of the widget.
- **pady:** This option will add padding to the widget vertically outside the border of the widget.
- **ipadx:** This option will add padding to the widget horizontally inside the border of the widget.
- **ipady:** This option will add padding to the widget vertically inside the border of the widget.
- **row:** This option will use a cell that will be identified with a given row whose default value is the first row with numeric value of 0.
- **rowspan:** This option will indicate the number of rows the widget occupies whose default value is 1.
- **sticky:** Whenever the cell is larger than the widget, an indication is required for the sides and cell corners to which the widget sticks. It may be a string concatenation of 0 or more of compass directions: M, E, S, W, NE, NW, SE, SW, and 0. The widget is centered in the cell with sticky = " and will take up the full cell area when NESW.

Kindly observe the given code:

```
from tkinter import *
myroot = Tk()

mybtn_col = Button(myroot, text="It is Column No. 4")
mybtn_col.grid(row = 0, column=4)

mybtn_colspan = Button(myroot, text="The colspan is of 4")
mybtn_colspan.grid(row = 1,columnspan=4)

mybtn_paddingx = Button(myroot, text="padx of 5 from outside wid-
get border")
mybtn_paddingx.grid(row = 2,padx=5)

mybtn_paddingy = Button(myroot, text="pady of 5 from outside wid-
get border")
mybtn_paddingy.grid(row = 3,pady=5)

mybtn_ipaddingx = Button(myroot, text="ipadx of 5 from inside wid-
get border")
mybtn_ipaddingx.grid(row = 4,ipadx=5)

mybtn_ipaddingy = Button(myroot, text="ipady of 15 from inside wid-
get border")
mybtn_ipaddingy.grid(row = 5,ipady=15)

mybtn_row = Button(myroot, text="It is Row No. 7")
mybtn_row.grid(row=7)

mybtn_rowspan = Button(myroot, text="It is Rowspan of 3")
mybtn_rowspan.grid(row = 8,rowspan=3)

mybtn_sticky = Button(myroot, text="Hey ! I am at North-West")
mybtn_sticky.grid(sticky=NW)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.36*:

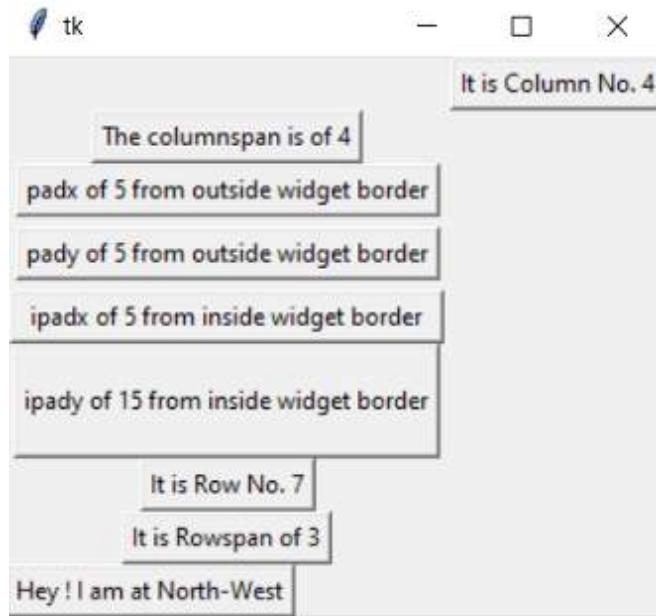


Figure 1.36: Output of Chap1_Example22.py

Note: The preceding code is covered in Program Name: **Chap1_Example22.py**

Note: It is not recommended to mix **grid()** and **pack()** in the same master window.

place()

It is simpler than the other 2 geometry managers. This geometry manager will organize the widgets in a specific position in the parent widget. It is placed at some specific position in the parent widget. It can be used with **pack()** as well as **grid()** method.

The syntax is

```
widget.place(options,...)
```

The options are:

- **anchor:** This option will indicate where the widget is anchored to and has compass directions as: N, S, E, W, NE, NW, SE, or SW, which are related to the sides and corners of the parent widget. The default compass direction is NW.
- **bordermode:** This option will indicate whether the widget border portion is included in the coordinate system or not. The default is INSIDE and the other is OUTSIDE.
- **relheight:** This option will specify the height as a float between 0.0 and 1.0, which is a fraction of the parent widget height, that is, the widget's height relates to the parent's widget height by ratio.
- **relwidth:** This option will specify the width as a float between 0.0 and 1.0, which is a fraction of the parent widget width, that is, the widget's width relates to the parent's widget width by ratio.
- **height:** This option will specify the widget height in pixels.
- **width:** This option will specify the widget width in pixels.
- **relx:** This option will specify the horizontal offset as a float between 0.0 and 1.0 which is a fraction of the parent widget width, that is, widget will be placed by x ratio relative to its parent.
- **rely:** This option will specify the vertical offset as a float between 0.0 and 1.0, which is a fraction of the parent widget height, that is, widget will be placed by y ratio relative to its parent.
- **x:** This option will specify the horizontal offset in pixels.
- **y:** This option will specify the vertical offset in pixels.

Note: In the above geometry manager, at least 2 options are required when invoked.

Just observe the following example:

```
from tkinter import *
myroot = Tk()
myroot.geometry("600x600")

mybtn_height = Button(myroot, text="60px high")
mybtn_height.place(height=60, x=300, y=300)

mybtn_width = Button(myroot, text="70px wide")
mybtn_width.place(width=70, x=400, y=400)

mybtn_relheight = Button(myroot, text="relheight of 0.7")
mybtn_relheight.place(relheight=0.7)

mybtn_rewidth= Button(myroot, text="relwidth of 0.4")
mybtn_rewidth.place(relwidth=0.4)

mybtn_relx=Button(myroot, text="relx of 0.5")
mybtn_relx.place(relx=0.5)

mybtn_rely=Button(myroot, text="rely of 0.8")
mybtn_rely.place(rely=0.8)

mybtn_x=Button(myroot, text="X = 500px")
mybtn_x.place(x=500)

mybtn_y=Button(myroot, text="Y = 520")
mybtn_y.place(y=520)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.37*:

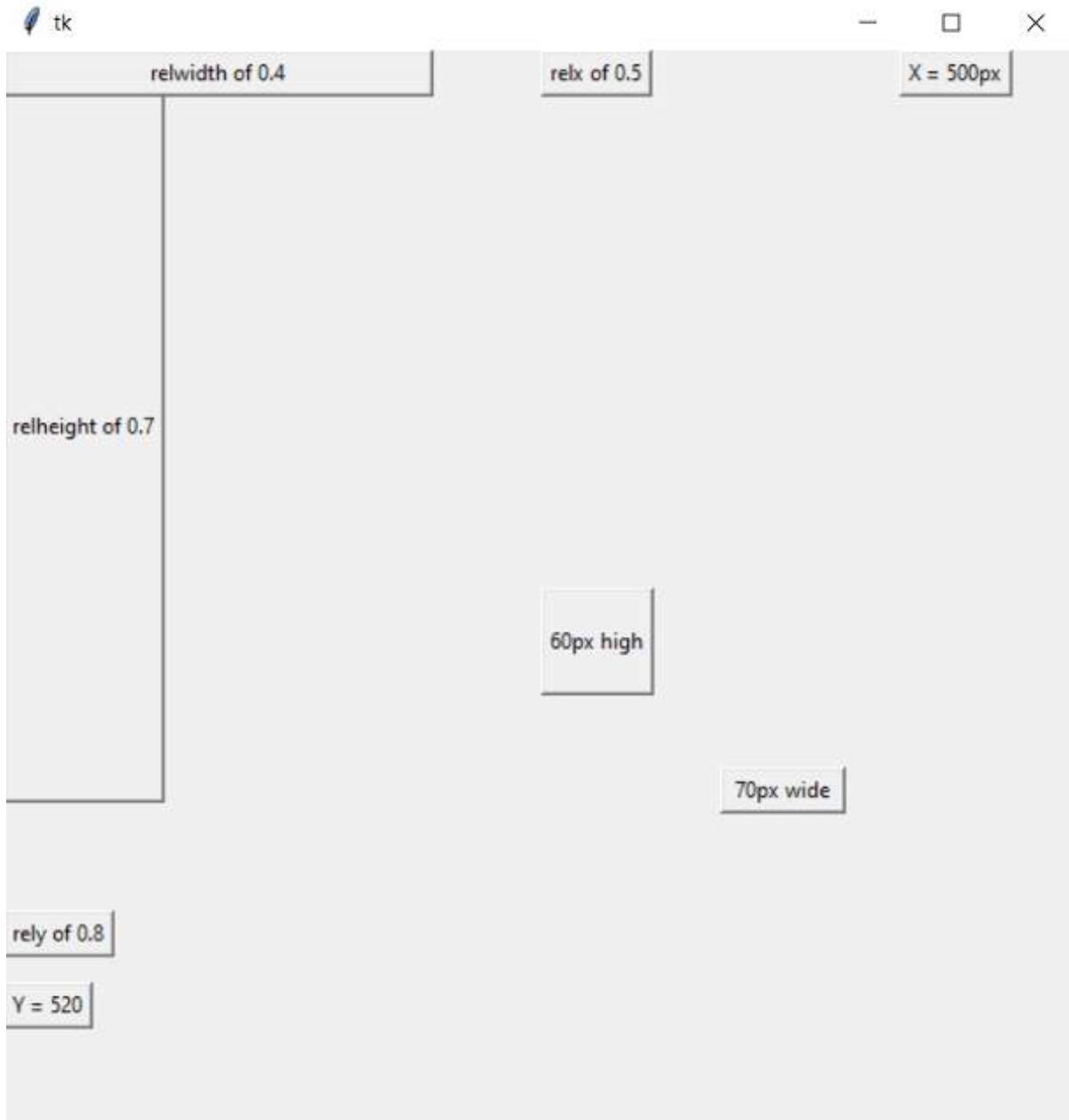


Figure 1.37: Output of Chap1_Example23.py

Note: The preceding code is covered in Program Name: Chap1_Example23.py

Geometry method in tkinter

Whenever there is a requirement to set tkinter window dimensions as well as set the main window position, then tkinter provides a geometry method, as follows:

```
from tkinter import *

# creating blank tkinter window
myroot = Tk()

myroot.geometry('300x150')

mybtn = Button(myroot, text = 'Python')
mybtn.pack(side = TOP, padx = 5, pady = 5)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.38*:



Figure 1.38: Output of Chap1_Example24.py

Note: The preceding code is covered in Program Name: **Chap1_Example24.py**

When we will run the preceding code, a fixed geometry of the tkinter window is created with dimensions ‘300x150’. Moreover, the tkinter window size will be changed but the screen position will be the same.

What if we are in need to change the position? We can do it by performing a little tweaking in the code as shown:

```
from tkinter import *

# creating blank tkinter window
myroot = Tk()

myroot.geometry('300x150+400+400')

mybtn = Button(myroot, text = 'Python')
mybtn.pack(side = TOP, padx = 5, pady = 5)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 1.39*:

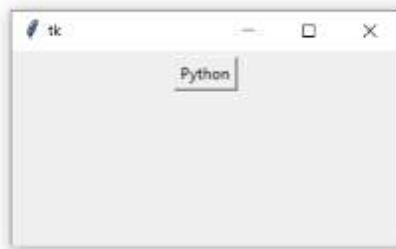


Figure 1.39: Output of Chap1_Example25.py

Note: The preceding code is covered in Program Name: Chap1_Example25.py

Now, when we will run the preceding code, then both the position and size are changed. We can see that the tkinter window is appearing in a different

position (400 shifted on each X and Y axis).

An important point to note is that the variable argument must be in the form of (variable1)x(variable2), otherwise an error will be raised.

Conclusion

In this chapter, we learned initially about basic GUI python program creation using tkinter library. Then we saw standard attributes of Python tkinter GUI such as dimensions, colors or fonts, along with various options with examples. We learned how to position the text with list of constants relative to the reference point using anchor attribute. We also saw an example of relief attribute, which brings 3-D simulated effects around the outside of the widget. The bitmap and cursor attribute were also explored with a crystal-clear example. Finally, towards the end of this chapter, we learned how to access tkinter widgets using inbuilt layout geometry managers viz pack, grid and place with syntax and its various options.

Points to remember

- Import tkinter library first for creation GUI applications using this library.
- Dimension set to an integer is assumed to be in pixels.
- Representation of colors in tkinter can be done in hexadecimal digits or by standard name.
- Font in tkinter can be accessed using a font object or by using a tuple.
- The text position relative to the reference point can be done using anchor attribute.
- 3-D simulated effects around the outside of the widget can be done using relief style.
- Widgets position using absolute positioning can be done using place geometry manager.
- Widgets in horizontal and vertical position is organized using pack geometry manager.

- Widgets in a 2-D grid can be positioned using grid geometry manager.

Questions

1. Write short note on GUI designing using the tkinter package.
2. Draw and explain the application level of Python.
3. Write a basic Python GUI program and explain its structure.
4. Explain any five standard attributes of Python tkinter GUI.
5. Explain in detail how the color is represented in tkinter, and explain any 5 options available for color representation.
6. Explain how fonts access is done in tkinter library of Python.
7. Draw and explain anchor constant in Python.
8. To have 3-D simulated effects around the outside of the widget, what style of a widget should be used? Explain in detail.
9. Write short notes on the following:
 - a. Bitmaps
 - b. Cursors
10. Explain Python tkinter Geometry Manager and its classes.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 2

Inbuilt Variable Classes for Python tkinter GUI Widgets

Introduction

This chapter will deal with inbuilt variable classes for Python tkinter GUI widgets and will also demonstrate the creation of a simple GUI Windows app using classes and object concepts.

We will also be learning how to access and set pre-defined variables subclassed from the tkinter variable class viz StringVar, IntVar, DoubleVar, and BooleanVar.

Structure

In this chapter, we will discuss the following topics:

- Inbuilt variable classes
- StringVar()
- BooleanVar()
- IntVar()
- DoubleVar()
- GUI creation using classes and objects

Objectives

By the end of this chapter, the reader will learn how to store data associated with widgets in tkinter, by using variable classes such as **StringVar()**, **BooleanVar()**, **IntVar()** and **DoubleVar()** for storing string, Boolean, integer and floating point values respectively. Finally, we shall see the importance of Tkinter's classes and objects which can be used to create GUIs, thus improving the code's organization, maintainability, and reusability, which also offers flexibility and aids in lowering the complexity of the code.

Inbuilt variable classes

A variable is needed with a wide variety of widgets. Whenever a user enters some text in the **Entry** widget or **Text** widget, a string variable is needed to track the written text. If there is a Checkbox widget, a Boolean variable is needed to track whether the user has checked or not. If the user requires some value in the **Spinbox** or **Slider** widget, then an integer variable is required to track the value. There is a Variable class in the tkinter which responds to changes in widget-specific variables. The commonly used pre-defined variables which are subclassed from the tkinter variable class are **StringVar**, **BooleanVar**, **IntVar**, and **DoubleVar**. We can associate one variable with more than one widget so that the same information can be displayed by more than one widget. Moreover, when the values are changed, the functions can be binded which are to be called. Methods such as **get()** and **set()** will be used to retrieve and set the values of these variables.

StringVar()

This variable will hold a string and the default value is an empty string, as follows:

```

from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window
# we should first know how to create a window if want to per-
form graphics coding.
# but output window will not be displayed right now

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

mystr = StringVar() # S1
print(type(mystr)) # S2
my_entry = Entry(myroot, font = ('Calibri',12),textvariable = mystr)
my_entry.pack()

def myshow():
    mydata = mystr.get() # S3
    print(mydata)
    mystr.set('') # S4

my_btn = Button(myroot, font = ('Cal-
ibri',12), text = 'Get Data!',command = myshow)
my_btn.pack()

myroot.mainloop()

```

Output initially and when text is written:

The output can be seen in *Figure 2.1*:

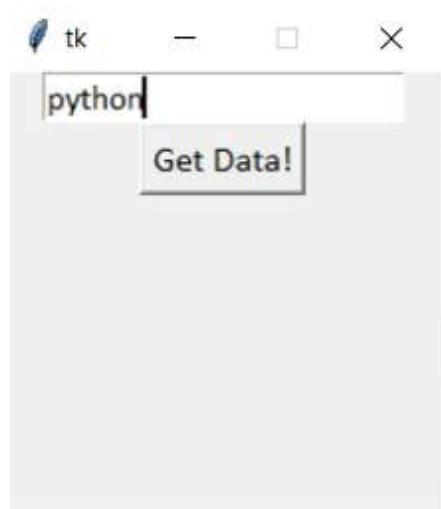


Figure 2.1: Initial Output

Output when Get Data! button is clicked:

The output can be seen in *Figure 2.2*:

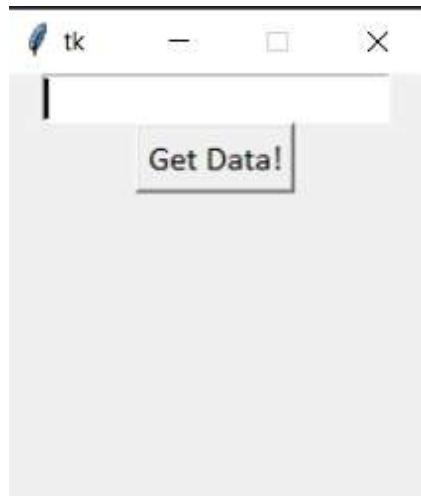


Figure 2.2: Output of Chap2_Example1.py

Note: The preceding code is covered in Program Name: **Chap2_Example1.py**

Output when a program is stopped or when 'x' is clicked:

The output is as follows:

```
<class 'tkinter.StringVar'>
python
```

In the preceding code, we have created two widgets: Entry and Button, in the parent widget and we are demonstrating the usage of the tkinter **StringVar()** type.

In S1, an instance of **StringVar()** type is created and is assigned to the Python **mystr** variable.

In S2, the type is <class 'tkinter.StringVar'>.

Then we are writing some text in the **Entry** widget and click the button Get Data!

In S3, we are using the variable **mystr** to get the value, saving it to a new variable named **mydata** and then displaying its value.

In S4, we are using the variable **mystr** to call the **set()** method on **StringVar()** and setting it to an empty string "".

An important point to observe is that we have used **textvariable** option, as it will associate a **tkinter** variable to the entry field contents. We will learn about other options associated when we will be dealing with the widgets in detail.

BooleanVar()

This variable will hold a Boolean and will return 1 for True and 0 for False, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry('200x100')

num1 = BooleanVar()
mystr = StringVar()

def mydatainsertion():
    if num1.get() == True:
        mystr.set('It is set to True')
    else:

        mystr.set('It is set to False')

myc1 = Checkbutton(myroot, variable = num1, font = ('Calibri',12), text = 'Python', command = mydatainsertion)
myc1.pack()

mye1 = Entry(myroot, width = 20, textvariable = mystr)
mye1.pack()

myroot.mainloop()
```

Output when checkbutton is checked:

The output can be seen in *Figure 2.3*:

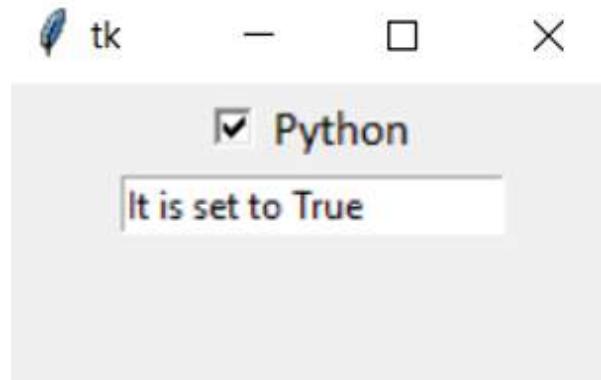


Figure 2.3: Output

Output when checkbutton is Unchecked:

Refer to *Figure 2.4*:

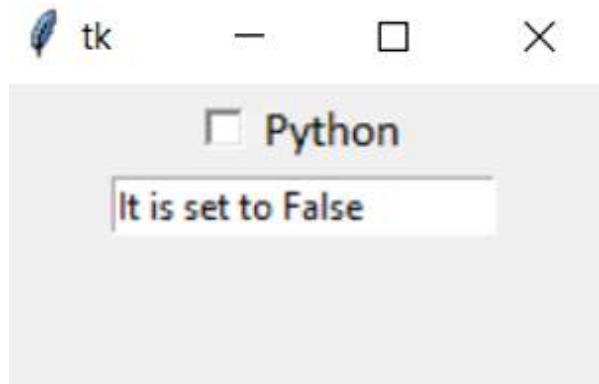


Figure 2.4: Output of Chap2_Example2.py

Note: The preceding code is covered in Program Name: Chap2_Example2.py

In the preceding code, when the user checks, the output will return True. Otherwise, on unchecking, it will return False.

IntVar()

This variable will hold an integer and the default value is 0, as shown below. If values are entered in fraction, the value will be truncated to an integer.

```

from tkinter import *
myroot = Tk()
myroot.geometry('200x200')
myroot.resizable(0,0)

myint = IntVar()
myint1 = IntVar()
myint2 = IntVar()

my_entry = Entry(myroot, font = ('Calibri',12),textvariable = myint)
my_entry.pack()

my_entry1 = Entry(myroot, font = ('Calibri',12),textvariable = myint1)
my_entry1.pack()

def mydisplay():
    mydata1 = myint.get()
    mydata2 = myint1.get()
    mydata3 = mydata1 * mydata2
    myint2.set(mydata3)

my_btn = Button(myroot, font = ('Calibri',12), text = 'Multi-
ply',command = mydisplay)
my_btn.pack()
my_entry2 = Entry(myroot, font = ('Calibri',12),textvari-
able = myint2)
my_entry2.pack()
my_entry2.configure(state = 'readonly')

myroot.mainloop()

```

Output when no data is entered:

Refer to *Figure 2.5*:

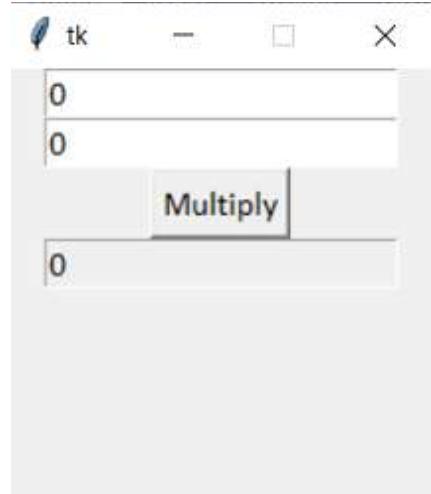


Figure 2.5: Output

Output when data is entered:

Refer to [*Figure 2.6*](#):

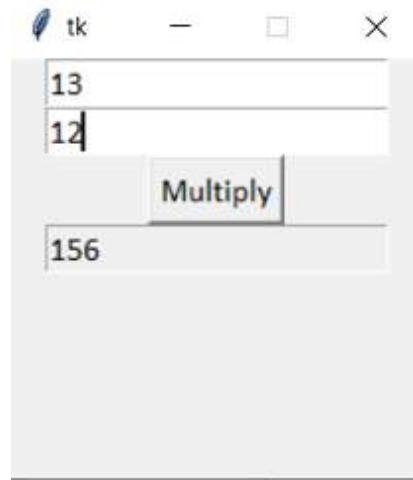


Figure 2.6: Output of *Chap2_Example3.py*

Note: The preceding code is covered in Program Name: **Chap2_Example3.py**

[**DoubleVar\(\)**](#)

This variable will hold a float and the default value is 0.0, as shown:

```
from tkinter import *

myroot = Tk()

myroot.geometry('200x200')
myroot.resizable(0,0)

myint = DoubleVar()
myint1 = DoubleVar()
myint2 = DoubleVar()

my_entry = Entry(myroot, font = ('Calibri',12),textvariable = myint)
my_entry.pack()

my_entry1 = Entry(myroot, font = ('Calibri',12),textvariable = myint1)
my_entry1.pack()

def mydisplay():
    mydata1 = myint.get()
    mydata2 = myint1.get()
    mydata3 = mydata1 - mydata2
    myint2.set(mydata3)

my_btn = Button(myroot, font = ('Calibri',12), text = 'Difference',command = mydisplay)
my_btn.pack()
my_entry2 = Entry(myroot, font = ('Calibri',12),textvariable = myint2)
my_entry2.pack()
my_entry2.configure(state = 'readonly')

myroot.mainloop()
```

Output when no data is entered:

Refer to *Figure 2.7*:

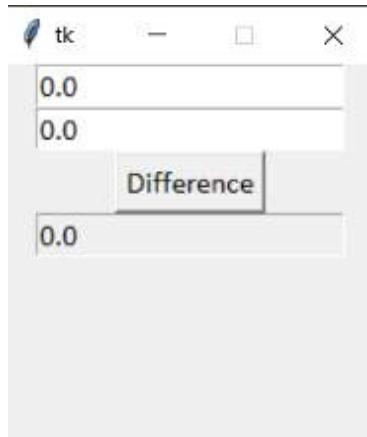


Figure 2.7: Output

Output when data is entered:

Refer to *Figure 2.8*:

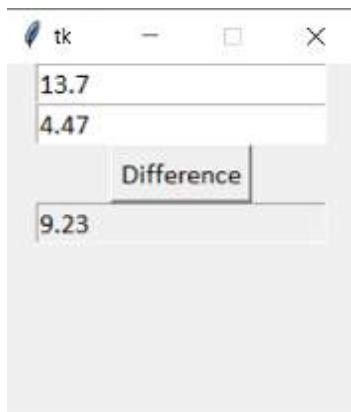


Figure 2.8: Output of Chap2_Example4.py

Note: The preceding code is covered in Program Name: Chap2_Example4.py

However, in the preceding code and the previous code of **IntVar()** variable class, we are entering numbers only. However, the user may enter anything in the **Entry** widget. So, we need to provide some checks such that the user may validate the **Entry** widget with numeric values only. We will learn when we will be dealing with widgets. For some time now, just observe the concept of variables.

GUI creation using classes and objects

As we have seen till now, it is very easy to create a basic GUI with a code of few lines only using **tkinter**. However, when the programs become complex, it is quite difficult to separate logic from each part. We need to make our code clean with an organized structure. Consider the following code:

```
from tkinter import *
from tkinter import messagebox
myroot = Tk()

def mydisplay():
    messagebox.showinfo('Message', "Python")

mybtn = Button(myroot, text="Display", command=mydisplay)
mybtn.pack(padx=20, pady=30)
myroot.mainloop()
```

Output:

Refer to *Figure 2.9*:

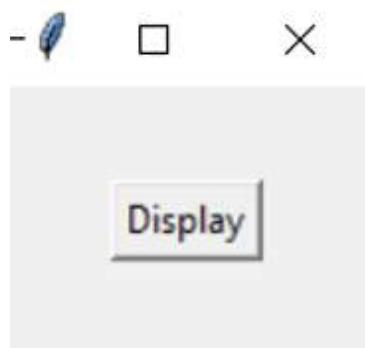


Figure 2.9: Output

Output on clicking Display button:

Refer to *Figure 2.10*:



Figure 2.10: Output of Chap2_Example5.py

Note: The preceding code is covered in Program Name: Chap2_Example5.py

In the preceding code, a main window is created having a **Button** widget. A message is being displayed as ‘Python’ when the **Display** button is clicked. The button widget is placed with a padding of 20px on the horizontal axis and 30px on the vertical axis. On executing the preceding code, we can verify that it is working as expected. We can see that all the variables are defined in the global namespace. However, it becomes and more difficult to reason about the parts usage when more widgets are added. Such type of issues can be addressed with basic OOP techniques.

Now, we will define a class that will wrap up our global variables:

```
from tkinter import *
from tkinter import messagebox

class MyBtn(Tk):
    def __init__(self): # constructor
        super().__init__() # for calling the constructor of
        self.mybtn = Button(self, text="Display", command=self.
        mydisplay)
        self.mybtn.pack(padx=20, pady=30)

    def mydisplay(self):
        messagebox.showinfo('Message', "Python")

if __name__ == "__main__":
    myroot = MyBtn() # making an object of MyBtn class
    myroot.mainloop()
```

Output:

Refer to *Figure 2.11*:

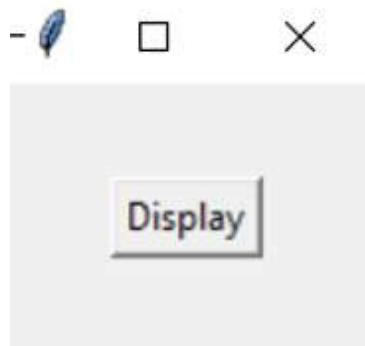


Figure 2.11: Output

Output on clicking Display button, output will be:

Refer to *Figure 2.12*:

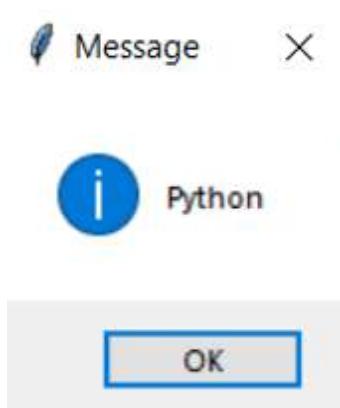


Figure 2.12: Output of Chap2_Example6.py

Note: The preceding code is covered in Program Name: Chap2_Example6.py

In the preceding code, we defined **MyBtn** class as a **Tk** subclass. To initialize the base class properly, the `__init__()` method of the **Tk** class is called with the built-in `super()` function. We have a reference to the **MyBtn** instance with the `self`-variable, and so the Button widget is added as an attribute of our class. The instantiation of the Button is separated from the callback, which gets executed when it is clicked. It is a common practice in executable Python scripts to use `if __name__ == "__main__"` which is an entry point of any program. To show the main window itself, we are calling the `mainloop` method inside `if __name__ == "__main__"` block. So, we can make any large Python code by taking note of the usage of classes and objects to create GUI applications.

The advantages of GUI creation using classes and objects in tkinter are as follows:

- **Encapsulation:** By using classes, we may encapsulate a GUI's functionality in a single class, simplifying the organization and maintenance of the code. Additionally, it lessens the complexity of the code and prevents naming disputes.
- **Reusability:** We can construct reusable code that can be utilised in other apps or sections of an application by defining classes and objects.
- **Modularity:** Classes and objects facilitate code modularization by making it simpler to divide different elements of a GUI, such as the

design and the functionality.

- **Inheritance:** Subclasses that are created through inheritance will take on the parent class's attributes and functions. When building similar GUI elements with somewhat differing functionality, this can save time and effort.
- **Flexibility:** Classes and objects give us the ability to tweak and update GUIs since they let us make code changes without having an impact on other areas of the program.
- **Code organization:** It helps us in organizing the code more clearly and makes it understandable by using classes and objects. When problems develop, it could also make it simpler to troubleshoot the code.

Conclusion

In this chapter, we learned different approaches of data storage associated with widgets in tkinter by using variable classes such as **StringVar()**, **BooleanVar()**, **IntVar()** and **DoubleVar()** for storing string, Boolean, integer and floating point values respectively with python code.

We have seen two approaches of displaying an application with and without usage of Tkinter's classes and objects for creating GUIs. Finally, we can say that the approach of using classes and objects for GUI creation helps us in improving the code's organization, maintainability, reusability and aids in lowering the complexity of the code.

Points to remember

- For storing and manipulating string values and for representing the text or widgets value like Entry and label, we may use **StringVar()** variable.
- For storing Boolean values and for representing the state of Checkbuttons or radiobuttons, we may use **BooleanVar()** variable.

- For storing integer values and for representing the widget's value like Spinbox or Scale, we may use **IntVar()** variable.
- For storing floating point values and for representing the widget's value like Spinbox or Scale, we may use **DoubleVar()** variable.
- GUI application creating in tkinter using classes and objects offers several advantages such as encapsulation, reusability, modularity, inheritance, flexibility and code organization.

Questions

1. Explain in detail about inbuilt variable classes for Python tkinter GUI widgets.
2. Explain the use of inbuilt variable classes for designing GUI in Python.
3. Explain the use of **StringVar()** in Python with a suitable example.
4. Which variable is used to return 1 for True and 0 for False values in Python for GUI? Explain in detail.
5. Explain **IntVar()** in detail with a suitable example.
6. Explain **DoubleVar()** in detail with a suitable example.
7. Write a short code for GUI Creation using Classes and Objects. What are its advantages?

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 3

Getting Insights of Button Widgets in tkinter

Introduction

A standard **Graphical User Interface (GUI)** element, which are basic building blocks of any GUI program are termed as widgets. We have observed till now that a top-level root window object contains different small window objects which are part of our developed GUI application. We put all the widgets in top-level window. We can have more than one top-level window, but can have only one root window. We shall see different widgets one by one now with starting from different widgets related to button in tkinter.

Structure

In this chapter, we will discuss the following topics:

- tkinter Button Widget
- tkinter Checkbutton Widget
- tkinter Radiobutton Widget
- tkinter OptionMenu Widget

Objectives

By the end of this chapter, the reader will learn about one of the most commonly used GUI widgets viz tkinter Button widget. We will be viewing the binding of events to the above widget with multiple examples and

different methods including lambda expressions. Next, we shall look into the Checkbutton widget, which will give the provision to the user to select more than one option. Users will also view different options to get the image in this widget. Next, we will see how to use the tkinter Radiobutton widget. Users will view various examples where exactly one of the predefined sets of options will be chosen. Last but not the least, we will explore the tkinter Option-Menu widget where the user views how a pop-menu and button widget will be created for an individual option selection from a list of options.

tkinter Button Widget

One of the most commonly used GUI widgets in tkinter Python is the Button widget. A method or a function can be associated with this widget when clicked. Different options can be set or reset as per need. This widget is generally used for interaction with the user.

The syntax is:

```
mybtn1= Button(myroot , options...)
```

where,

myroot is the parent window.

Some of the lists of options that can be used as key-value pairs and are separated by commas are activebackground, activeforeground, bg, bd, command, fg, font, height, highlightcolor, justify, image, padx, pady, state, relief, width, wraplength, and underline. We have seen most of the options but some undiscussed options are as follows:

- **command:** This option will call the function or method whenever the button is clicked. So, by using the command option, we are adding functionality to the button.
- **justify:** This option will define the alignment of multiple lines of text with respect to each other. The default is CENTER and the other is LEFT or RIGHT.

Refer to the following code:

```
from tkinter import *
from tkinter import messagebox
```

```
class MyJustify(Tk):
    def __init__(self):
        super().__init__()
        self.title('Jusify in Button')

    def mycenterjustify():
        messagebox.showinfo('Justify','Justify CENTER')

    def myleftjustify():
        messagebox.showinfo('Justify','Justify LEFT')

    def myrightjustify():
        messagebox.showinfo('Justify','Justify RIGHT')

    # default justify is CENTER
    mybtn1= Button(self, text = 'JUSTIFY\nCENTER\nCENTER CENTER',bd = 3, relief = 'groove',
                    font = ('Helvetica',10), width = 20, height = 3, command = mycenterjustify)
    mybtn1.pack(pady= 10, side = BOTTOM)

    mybtn2= Button(self, text = 'JUSTIFY\nLEFT\nLEFT LEFT',bd = 3, relief = 'groove',
                    font = ('Helvetica',10), justify = LEFT, width = 20, height = 3, command = myleftjustify)
    mybtn2.pack(pady= 10, side = RIGHT)

    mybtn3= Button(self, text = 'JUSTIFY\nRIGHT\nRIGHT RIGHT',bd = 2,font = ('Helvetica',10),
                    justify = RIGHT, width = 20, height = 3, command = myrightjustify)
    mybtn3.pack(side = TOP)

if __name__ == "__main__":
    myroot = MyJustify()
    myroot.geometry('350x350')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 3.1*:

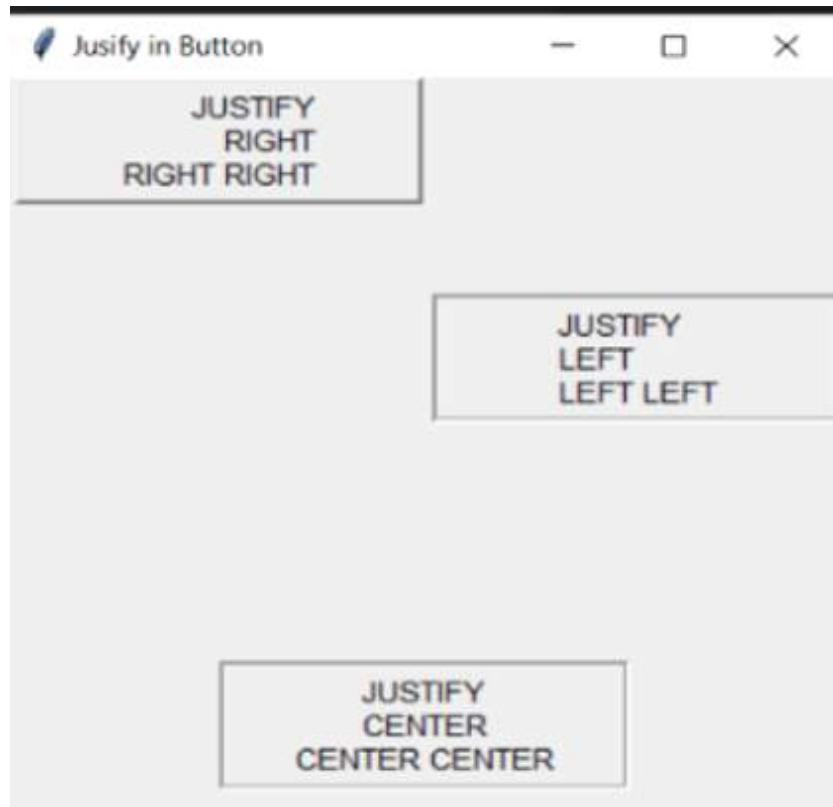


Figure 3.1: Output

Output when JUSTIFY LEFT button is clicked:

Refer to *Figure 3.2*:

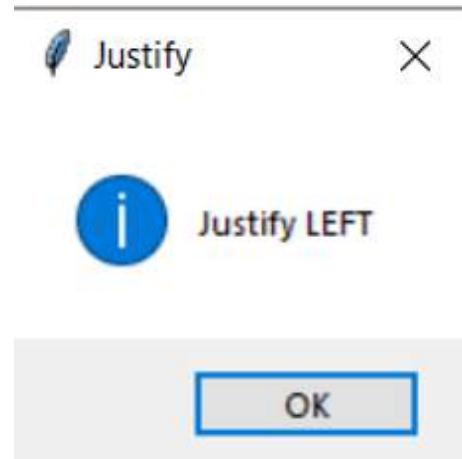


Figure 3.2: Output

Output when JUSTIFY RIGHT button is clicked:

Refer to *Figure 3.3*:

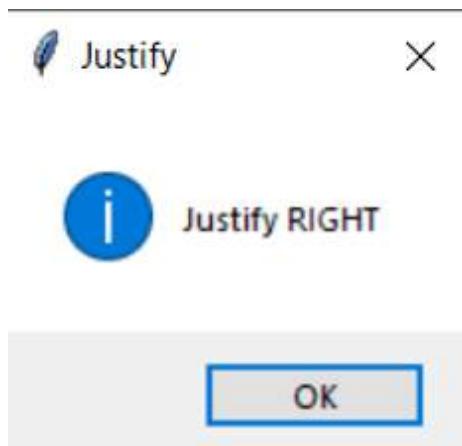


Figure 3.3: Output

Output when JUSTIFY CENTER button is clicked:

Refer to *Figure 3.4*:



Figure 3.4: Output

Note: The preceding code is covered in Program Name: Chap3_Example1.py

Let us go over the various options now:

image: This option will set the image to be displayed on the button instead of text.

state: This option, by default, is NORMAL. When this option is set to DISABLED, the button becomes unresponsive and will gray out the button.

Note: User may use 'Add-icon.png' or any other icon file as per need in their active directory. Here, we have added 'Add-icon.png' file as a reference.

Refer to the following code:

```

from tkinter import *

class MyBtnImage(Frame):
    def __init__(self, root = None):
        Frame.__init__(self, root)
        self.root = root
        self.myphoto = PhotoImage(file = 'Add-icon.png')
    def myclick():
        self.mybtn1['state'] = DISABLED
        self.mybtn1 = Button(self.root,image = self.
myphoto, command = myclick)
        self.mybtn1.pack(padx = 10, pady = 10)

if __name__ == "__main__":
    myroot = Tk()
    myobj = MyBtnImage(myroot)
    myroot.title('Image using Button')
    myroot.geometry('400x100')
    myroot.mainloop()

```

Output:

Refer to *Figure 3.5*:



Figure 3.5: Output

Output when button is clicked:

Refer to *Figure 3.6*:



Figure 3.6: Output

Note: The preceding code is covered in Program Name: Chap3_Example2.py

In the above code, we have added an image with a ‘+’ symbol to a button and when it is clicked, the button becomes disabled.

An important point to note is that only the image will appear on the screen when both text and image are given on the Button, as the text will be dominated by the image. But what if we want to display both text and image on the button? In such cases, we will be using the *compound* option in the button widget. Suppose we want the image to appear at bottom of the button. In this case, compound = BOTTOM.

Similarly, when compound = LEFT, the image will appear on the left side of the button widget; when compound = RIGHT, the image will appear on the right side of the button widget, and when compound = TOP, the image will appear on the top side of the button widget.

Let us see the code for a better understanding:

```
from tkinter import *

class MyBtnTextWithImage(Frame):
    def __init__(self, root = None):
        Frame.__init__(self, root)
        self.root = root
        self.myphoto = PhotoImage(file = 'Add-icon.png')
        self.mybtn1 = Button(self.root,image = self.
myphoto, text = 'Hello', compound = LEFT)
        self.mybtn1.pack(padx = 10, pady = 10)
        self.mybtn2 = Button(self.root,image = self.
myphoto, text = 'Hello', compound = RIGHT)
        self.mybtn2.pack(padx = 10, pady = 10)
        self.mybtn3 = Button(self.root,image = self.
myphoto, text = 'Hello', compound = TOP)
        self.mybtn3.pack(padx = 10, pady = 10)
        self.mybtn4 = Button(self.root,image = self.
myphoto, text = 'Hello', compound = BOTTOM)
        self.mybtn4.pack(padx = 10, pady = 10)

if __name__ == "__main__":
    myroot = Tk()
    myobj = MyBtnTextWithImage(myroot)
    myroot.title('Image using Button')
    myroot.geometry('400x200')
    myroot.mainloop()
```

Output:

Refer to *Figure 3.7*:

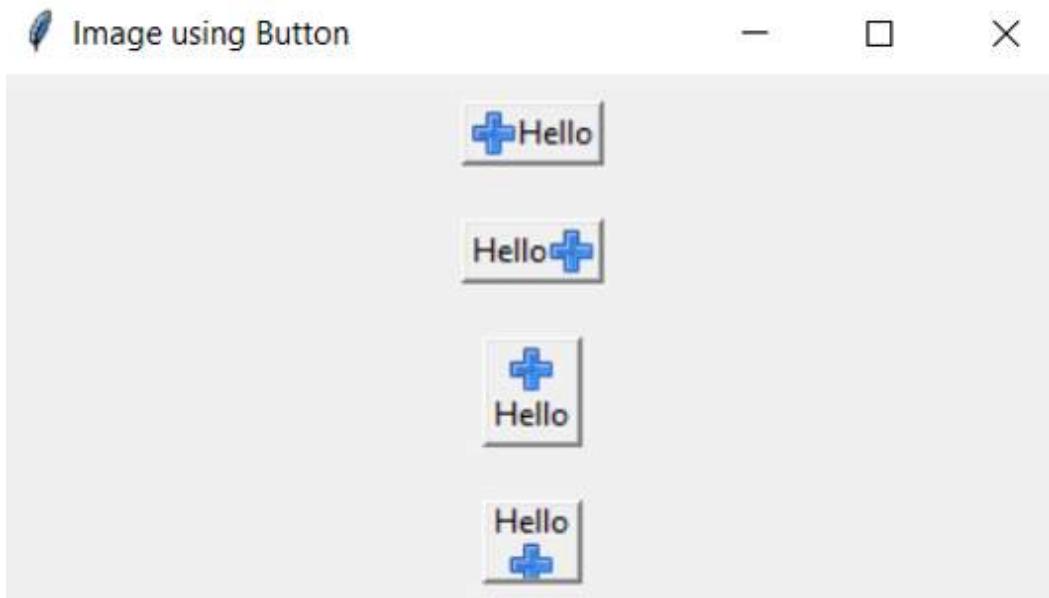


Figure 3.7: Output of Chap3_Example3.py

Note: The preceding code is covered in Program Name: Chap3_Example3.py

Now, we will discuss events and bindings before moving on to the next widgets.

Events and bindings

We have seen in all the examples till now that until and unless we press the 'X' mark of a parent widget, a tkinter code is spending most of the time inside an event loop (the mainloop method). We can see events such as mouse click, focusin, focusout, keypress events, and so on. In order to bind an event with a function, a **bind()** function is used, which is included in all the widgets whose syntax is given as:

```
widget.bind(event,handler, add = '')
```

Here, an event can be attached binding to a widget using the bind method.

The first argument event is a representative string that must be in the following format:

- <modifier-type-detail>: Here, the modifier and detail sections are optional and the only mandatory part is the type section which

represents the event type to listen for. We will discuss each section one by one.

- **event modifiers:** They can change the circumstances in which an event's handler is activated and is an optional component for creating an event binding. We should note that most of the event handlers are platform specific and will not work on all platforms.
- **Shift:** While the event is occurring, the shift button needs to be pressed.
- **Control:** While the event is occurring, the Control button needs to be pressed.
- **Alt:** While the event is occurring, the Alt button needs to be pressed.
- **Lock:** When the event occurs, the caps lock button needs to be activated.
- **Double:** The event will be happening twice in quick succession, say double-click.
- **Triple:** The event will be happening thrice in quick succession.
- **Quadruple:** The event will be happening four times in quick succession.

event type

The different event types in tkinter are as follows:

- **ButtonPress or Button:** An event will be generated or activated when a mouse button has been clicked. Event <Button-1> defines the left mouse button, Event <Button-2> defines the middle button, Event <Button-3> defines the right mouse button, Event <Button-4> defines scroll-up on mice with wheel support, Event <Button-5> defines scroll-down on mice with wheel support.
- **ButtonRelease:** An event will be generated or activated when a mouse button has been released. Events <ButtonRelease-1>, <ButtonRelease-2> and <ButtonRelease-3> will specify the left, middle, or right mouse button.

- **Keypress or Key:** An event will be generated or activated when a keyboard button has been pressed.
- **KeyRelease:** An event will be generated or activated when a keyboard button is released.
- **Motion:** An event will be generated or activated when the mouse cursor is moved across the widget. Events <B1-Motion>, <B2-Motion> and <B3-Motion> will specify the left, middle, or right mouse button. The mouse pointer's current position will be provided in the x and y members of the event object passed to the callback, that is, event.x, event.y.
- **Enter:** An event will be generated or activated when the mouse cursor enters the widget.
- **Leave:** An event will be generated or activated when the mouse cursor leaves the widget.
- **FocusIn:** An event will be generated or activated when the widget gains the input focus.
- **FocusOut:** An event will be generated or activated when the widget loses the input focus.
- **Configure:** An event will be generated or activated when the widget configurations are changed such as width, height, or border width being adjusted by the user, and so on.
- **Mousewheel:** An event will be generated or activated when the mouse wheel is scrolled.
- **Event details:** It is an optional section and will serve as either a mouse button or a certain key on the keyboard being pressed.
- **For keyboard events:** The keyboard event detail is captured such that each key pressed on the keyboard will be represented by a key symbol or key letter itself. The ASCII value of the specific key is given for triggering the event when using Key, KeyPress, or KeyRelease.
- **For mouse events:** The mouse event detail is captured such that numeric detail from 1 to 5 will represent the specific mouse button we

wish to have to handle the trigger.

The second argument is a handler which represents the function name to call (callback function) when the event occurs. It takes an event parameter.

The attributes for the mouse events are as follows:

- **x and y**: It will return the x and y coordinate mouse position in pixels where events such as Buttons occur.
- **x_root and y_root**: It is similar to x and y but is relative to the upper-left screen corner.
- **num**: It returns the mouse button number.

The attributes for the keyboard events are:

- **char**: It is for keyboard events only and pressed character as a string.
- **keysym**: It is for keyboard events only and pressed key symbol.
- **keycode**: It is for keyboard events only and pressed key code.

As the event handler, the function included will be passed as an event object which will describe the events and include details about the event which was triggered. So, a parameter is to be included which will be assigned to this object in the function.

The third parameter, which can be None, is added to replace the callback if there was a previous binding or ‘+’ to preserve the old ones and add the callback.

Now, if we want to bind an event to a widget instance, then it is called *instance-level binding*.

There are times when it is needed to bind events to an entire application, which is called application-level binding where the same binding is used across all the windows and application widgets as long as any one application window is in focus. The syntax of application-level binding is:

`widget.bind_all(event,callback)`

For example:

```
myroot.bind_all('<F1>', show_help)
```

Here, if we press the *F1* key and then the **show_help** callback will always trigger, irrespective of any widget the focus as long as the application is under active focus.

Another is *class-level binding* where the events can be bound at a particular class level. The syntax of class-level binding is:

```
widget.bind_class(classname, event,callback)
```

For example,

```
mye1.bind_class('<Entry>', '<Control-C>', copy)
```

Here, all the Entry widgets will be bound to the **<Control-C>** event which would call a method called 'copy (event)'.

We shall see some examples related to buttons, events, and bindings for a better understanding of the concepts.

```

from tkinter import * # importing module
class MyLeftRightMouseClick(Tk):
    def __init__(self):
        super().__init__()
        self.title('Button Left and Right click')

    def mycall(event):
        print('Left Clicked')

    def mycallme(event):
        print('Right Clicked')

    self.
myb1 = Button(self, text = 'LeftClick', font = ('Calibri',15))
    self.myb1.bind('<Button-1>', mycall) # Left click
    self.myb1.pack(pady = 10) # for displaying the button

    self.
myb2 = Button(self, text = 'RightClick', font = ('Calibri',15))
    self.myb2.bind('<Button-3>', mycallme) # Right click
    self.myb2.pack(pady = 10) # for displaying the button

if __name__ == "__main__":
    myroot = MyLeftRightMouseClick()
    myroot.geometry('350x150')
    myroot.
mainloop() # display window until we press the close button

```

Initial output:

Refer to *Figure 3.8*:



Figure 3.8: Output

Output when LeftClick button is clicked using the left side of the mouse:

Refer to [Figure 3.9](#):

```
$ python mypythonguiprog.py  
Left Clicked
```

Figure 3.9: Output

Output when RightClick button is clicked using the mouse right side after clicking the mouse left side:

Refer to [Figure 3.10](#):

```
$ python mypythonguiprog.py  
Left Clicked  
Right Clicked
```

Figure 3.10: Output

Note: The preceding code is covered in Program Name: Chap3_Example4.py

In this code, we bind the button widget to the event <'Button-1'> which corresponds to the mouse left click. Whenever this event will occur, the

method **mycall** will be called, which will be passing an object instance as its argument. The method *mycall(event)* will take an event object generated by the event as its argument. So, when the left button of the mouse is clicked on LeftClick button, then the Left Clicked message will be displayed.

We bind the button widget to the event <'**Button-3**> which corresponds to the mouse's Right click. Whenever this event will occur, the method **mycallme** will be called, which will be passing an object instance as its argument. The method *mycallme(event)* will take an event object generated by the event as its argument. So, when the right button of the mouse is clicked on RightClick button, then the Right Clicked message will be displayed.

We can display some messages in the command prompt via event handling:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('370x100') # but can be resized to any pixel un-
til we are using myroot.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.
myroot.title('event handling through Cmd prompt')

def mydisplay():
    print("Clicked !!!")

mytk_button1 = Button(myroot, text = 'Login',font = ('Cal-
ibri',15),fg = 'Blue',command = mydisplay)
mytk_button1.pack()

myroot.mainloop()
```

Output initial:

Refer to *Figure 3.11*:



Figure 3.11: Output

Output when the Button is clicked and the message is displayed on the command prompt

Refer to [*Figure 3.12:*](#)

```
$ python mypythonguiprog.py  
Clicked !!!
```

Figure 3.12: Output

Note: The preceding code is covered in Program Name: Chap3_Example5.py

We can change the background color using **config()** method. So, we will be looking at event handling through **config** method.

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('300x300') # but can be resized to any pixel until we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larger or smaller.

def myshow1():
    myroot.configure(background = 'LightBlue')

mytk_button1 = Button(myroot, text = 'Change Background color', font = ('Calibri',15),fg = 'Blue')
mytk_button1.config(command = myshow1)
mytk_button1.pack()

myroot.mainloop()
```

Output initially:

Refer to *Figure 3.13*:

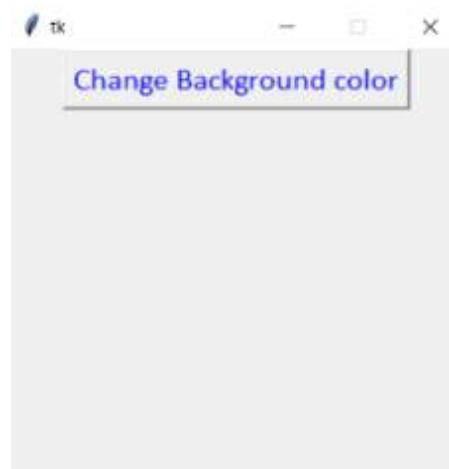


Figure 3.13: Output

Output when the Button is clicked and the message is displayed on the command prompt:

Refer to *Figure 3.14*:

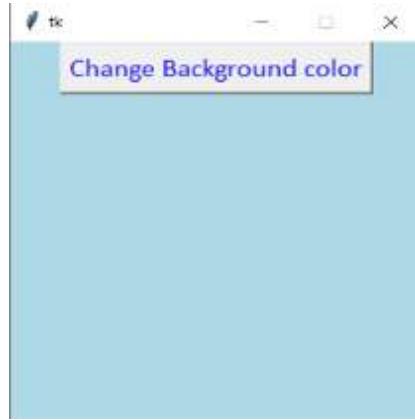


Figure 3.14: Output

Note: The preceding code is covered in Program Name: Chap3_Example6.py

Now, we will be viewing event handling using the bind method:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel until we are using myroot.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larger or smaller.

def myshow1(e):
    myroot.configure(background = 'LightBlue')

def myshow2(e):
    myroot.configure(background = 'LightGreen')

def myshow3(e):
    myroot.configure(background = 'LightPink')

mytk_btn1 = Button(myroot, text = 'Background color',font = ('Calibri',15),fg = 'Blue')

mytk_btn1.bind('<Button-1>',myshow1) # Left Key of mouse
mytk_btn1.bind('<Button-2>',myshow2) # Wheel Key of mouse
mytk_btn1.bind('<Button-3>',myshow3) # Right Key of mouse
mytk_btn1.pack()

myroot.mainloop()
```

Output initially:

Refer to *Figure 3.15*:

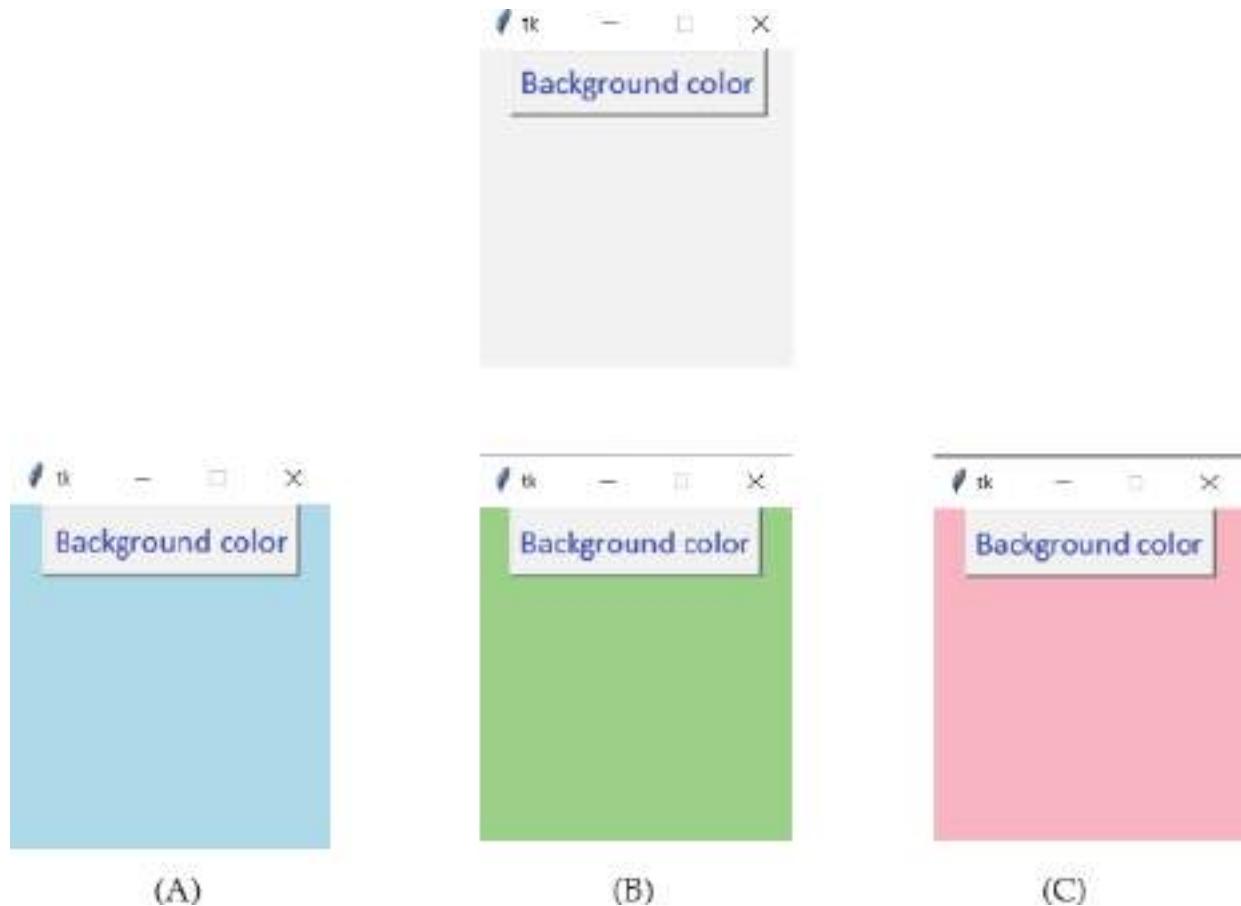


Figure 3.15: Output

Here,

- A) Output when the left key of the mouse is clicked.
- B) Output when the wheel key of the mouse is clicked.
- C) Output when the right key of the mouse is clicked.

Note: The preceding code is covered in Program Name: Chap3_Example7.py

Now, we shall see event handling using a lambda expression. A lambda expression in Python is a tool for writing anonymous functions. Without the requirement for a formal function definition, it enables you to build short, inline functions. A lambda expression can have a single expression but any number of parameters.

The advantages of lambda expressions over conventional methods are as follows:

- **Conciseness:** By allowing the user to write functions on a single line using lambda expressions, the code is made shorter and simpler to read.
- **Readability:** By grouping together relevant functionality, lambda expressions can be utilized to build tiny, straightforward functions inline, improving the readability of user's code.
- **Avoiding function definition:** Using lambda expressions eliminates the need to declare a function individually, which is advantageous when user only require a function once and do not want to clutter the code with many definitions.
- **Function as argument:** When working with collections, lambda expressions can be supplied as arguments to higher-order functions like `map()`, `filter()`, and `reduce()`. This enables more concise and expressive code.

We will be viewing the output of the previous example using lambda expressions:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

#using lambda expression
myshow1 = lambda e: myroot.configure(background = 'LightBlue')
myshow2 = lambda e: myroot.configure(background = 'LightGreen')
myshow3 = lambda e: myroot.configure(background = 'LightPink')

mytk_btn1 = Button(myroot, text = 'Background color',font = ('Cal-
ibri',15),fg = 'Blue')
mytk_btn1.bind('<Button-1>',myshow1) # Left Key of mouse
mytk_btn1.bind('<Button-2>',myshow2) # Wheel Key of mouse
mytk_btn1.bind('<Button-3>',myshow3) # Right Key of mouse
mytk_btn1.pack()

myroot.mainloop()
```

Note: The preceding code is covered in Program Name: Chap3_Example8.py

The output will be the same as the previous one. We have used lambda expressions instead of functions.

We can even use lambda expressions with **bind()** method:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

mytk_btn1 = Button(myroot, text = 'Background color',font = ('Cal-
ibri',15),fg = 'Blue')
mytk_btn1.bind('<Button-1>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # Left Key of mouse
mytk_btn1.bind('<Button-2>',lambda e: myroot.configure(back-
ground = 'LightGreen')) # Wheel Key of mouse
mytk_btn1.bind('<Button-3>',lambda e: myroot.configure(back-
ground = 'LightPink')) # Right Key of mouse
mytk_btn1.pack()

myroot.mainloop()
```

Note: The preceding code is covered in Program Name: Chap3_Example9.py

The output is still the same.

Now, suppose we want the same output but on mouse double click. In that case, we need to do a small modification, such that instead of <Button-1>, we will use <Double-1> and so on which is displayed as follows:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

mytk_btn1 = Button(myroot, text = 'Background color',font = ('Cal-
ibri',15),fg = 'Blue')
mytk_btn1.bind('<Double-1>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # Left Key of mouse on double click
mytk_btn1.bind('<Double-2>',lambda e: myroot.configure(back-
ground = 'LightGreen')) # Wheel Key of mouse on double click
mytk_btn1.bind('<Double-3>',lambda e: myroot.configure(back-
ground = 'LightPink')) # Right Key of mouse on double click
mytk_btn1.pack()

myroot.mainloop()
```

Note: The preceding code is covered in Program Name: Chap3_Example10.py

Here, we will get the same desired result, but on double clicking of the mouse. When the left key of the mouse is double-clicked, we will get **LightBlue** color. When the wheel key of the mouse is double-clicked, we will get **LightGreen** color and when the right key of the mouse is double-clicked, we will get **LightPink** color.

We can even change the background color by using Enter and Leave events on the Button widget, as follows:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

mytk_btn1 = Button(myroot, text = 'Background color',font = ('Cal-
ibri',15),fg = 'Blue')
mytk_btn1.bind('<Enter>',lambda e: myroot.configure(back-
ground = 'LightBlue'))
mytk_btn1.bind('<Leave>',lambda e: myroot.configure(back-
ground = 'LightGreen'))
mytk_btn1.pack()

myroot.mainloop()
```

Output when mouse pointer is entering into Button widget:

Refer to *Figure 3.16*:

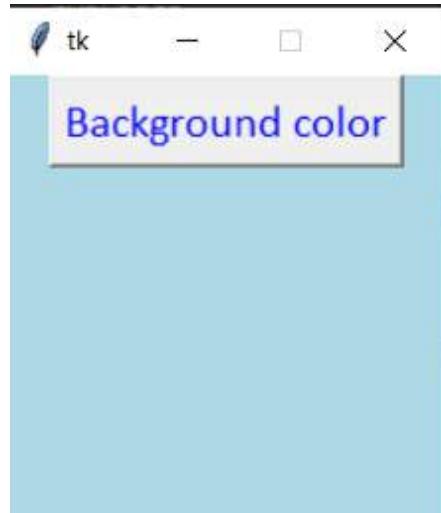


Figure 3.16: Output

Output when mouse pointer is leaving from Button widget:

Refer to *Figure 3.17*:

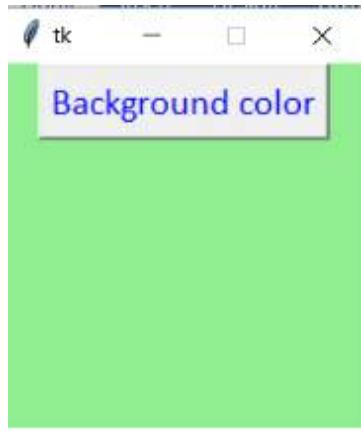


Figure 3.17: Output

Note: The preceding code is covered in Program Name: Chap3_Example11.py

In the preceding code, when the mouse pointer is entering into the button widget, the background color of the window will be LightBlue and when the mouse pointer is leaving the button widget, the background color of the window will be LightGreen.

Now, we shall see event handling on pressing alphabet keys:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

myroot.bind('<Key-a>',lambda e: myroot.configure(background = 'Light-
Blue')) # on pressing key 'a'
myroot.bind('<Key-b>',lambda e: myroot.configure(background = 'Light-
Green')) # on pressing key 'b'
myroot.bind('<Key-c>',lambda e: myroot.configure(background = 'Light-
Pink')) # on pressing key 'c'

myroot.mainloop()
```

Output:

Refer to *Figure 3.18*:



Figure 3.18: Output

Here,

A) On pressing Key-a, the window color changes to LightBlue.

B) On pressing Key-b, the window color changes to LightGreen.

C) On pressing Key-c, the window color changes to LightPink.

Note: The preceding code is covered in Program Name: Chap3_Example12.py

We can also perform event handling by pressing special keys such as *F1*, *F2*, *F3*, *Delete* and so on:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

myroot.bind('<F1>',lambda e: myroot.configure(background = 'Light-
Blue')) # on key pressing Fn+F1 in laptop
myroot.bind('<F2>',lambda e: myroot.configure(background = 'Light-
Green')) # on key pressing Fn+F2 in laptop
myroot.bind('<F3>',lambda e: myroot.configure(background = 'Light-
Pink')) # on key pressing Fn+F3 in laptop
myroot.bind('<Delete>',lambda e: myroot.configure(back-
ground = 'LightYellow')) # on key pressing Delete

myroot.mainloop()
```

Output:

Refer to *Figure 3.19*:

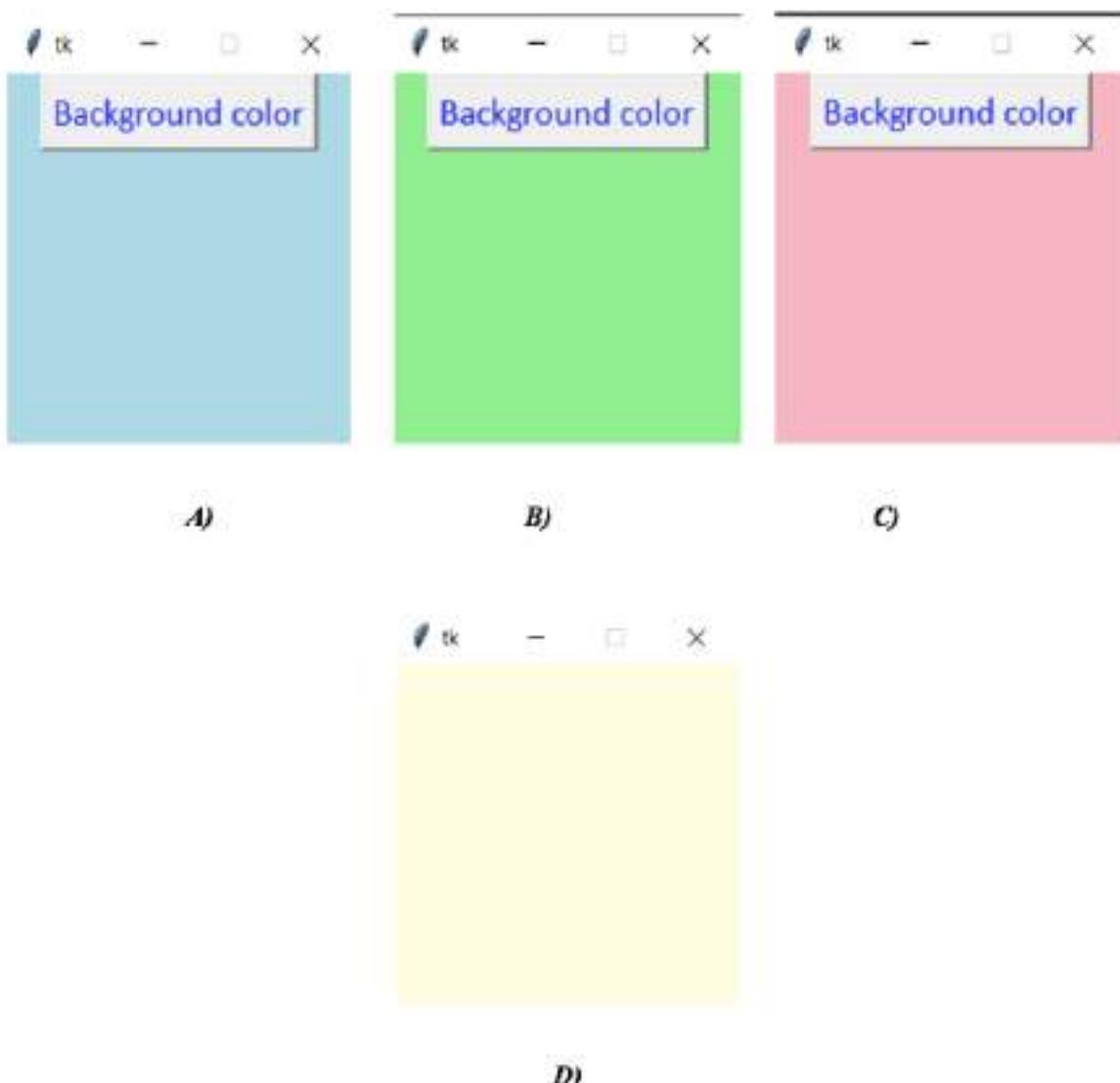


Figure 3.19: Output

Here,

- A) On pressing Key-*F1*, the window color changes to LightBlue.
- B) On pressing Key-*F2*, the window color changes to LightGreen.
- C) On pressing Key-*F3*, the window color changes to LightPink.
- D) On pressing *Delete*, the window color changes to LightYellow.

Note: The preceding code is covered in Program Name: Chap3_Example13.py

Event handling can also be performed by using number keys as follows:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

myroot.bind('1',lambda e: myroot.configure(background = 'Light-
Blue')) # on key pressing 1 in laptop
myroot.bind('2',lambda e: myroot.configure(background = 'Light-
Green')) # on key pressing 2 in laptop
myroot.bind('3',lambda e: myroot.configure(background = 'Light-
Pink')) # on key pressing 3 in laptop

myroot.mainloop()
```

Output:

Refer to *Figure 3.20*:

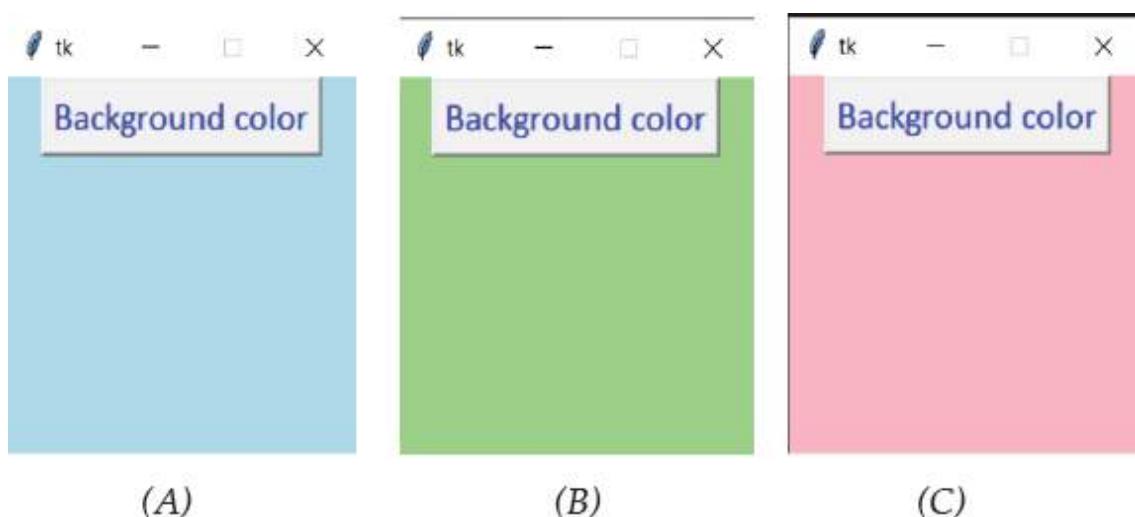


Figure 3.20: Output

Here,

- A) On pressing 1, the window color changes to LightBlue.
- B) On pressing 2, the window color changes to LightGreen.
- C) On pressing 3, the window color changes to LightPink.

Note: The preceding code is covered in Program Name: Chap3_Example14.py

We can also perform event handling by pressing *Shift*, *Alt*, and *Ctrl*:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

myroot.bind('<Shift-Up>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # on key pressing Shift-Up
myroot.bind('<Shift-Down>',lambda e: myroot.configure(back-
ground = 'LightGreen')) # on key pressing Shift-Down
myroot.bind('<Shift-Left>',lambda e: myroot.configure(back-
ground = 'LightPink')) # on key pressing Shift-Left
myroot.bind('<Shift-Right>',lambda e: myroot.configure(back-
ground = 'LightYellow')) # on key pressing Shift-Right

myroot.bind('<Alt-Up>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # on key pressing Alt-Up
myroot.bind('<Alt-Down>',lambda e: myroot.configure(back-
ground = 'LightGreen')) # on key pressing Alt-Down
myroot.bind('<Alt-Left>',lambda e: myroot.configure(back-
ground = 'LightPink')) # on key pressing Alt-Left
myroot.bind('<Alt-Right>',lambda e: myroot.configure(back-
ground = 'LightYellow')) # on key pressing Alt-Right

myroot.bind('<Control-Up>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # on key pressing Control-Up
myroot.bind('<Control-Down>',lambda e: myroot.configure(back-
ground = 'LightGreen')) # on key pressing Control-Down
myroot.bind('<Control-Left>',lambda e: myroot.configure(back-
ground = 'LightPink')) # on key pressing Control-Left
myroot.bind('<Control-Right>',lambda e: myroot.configure(back-
ground = 'LightYellow')) # on key pressing Control-Right

myroot.mainloop()
```

Output:

Refer to *Figure 3.21*:

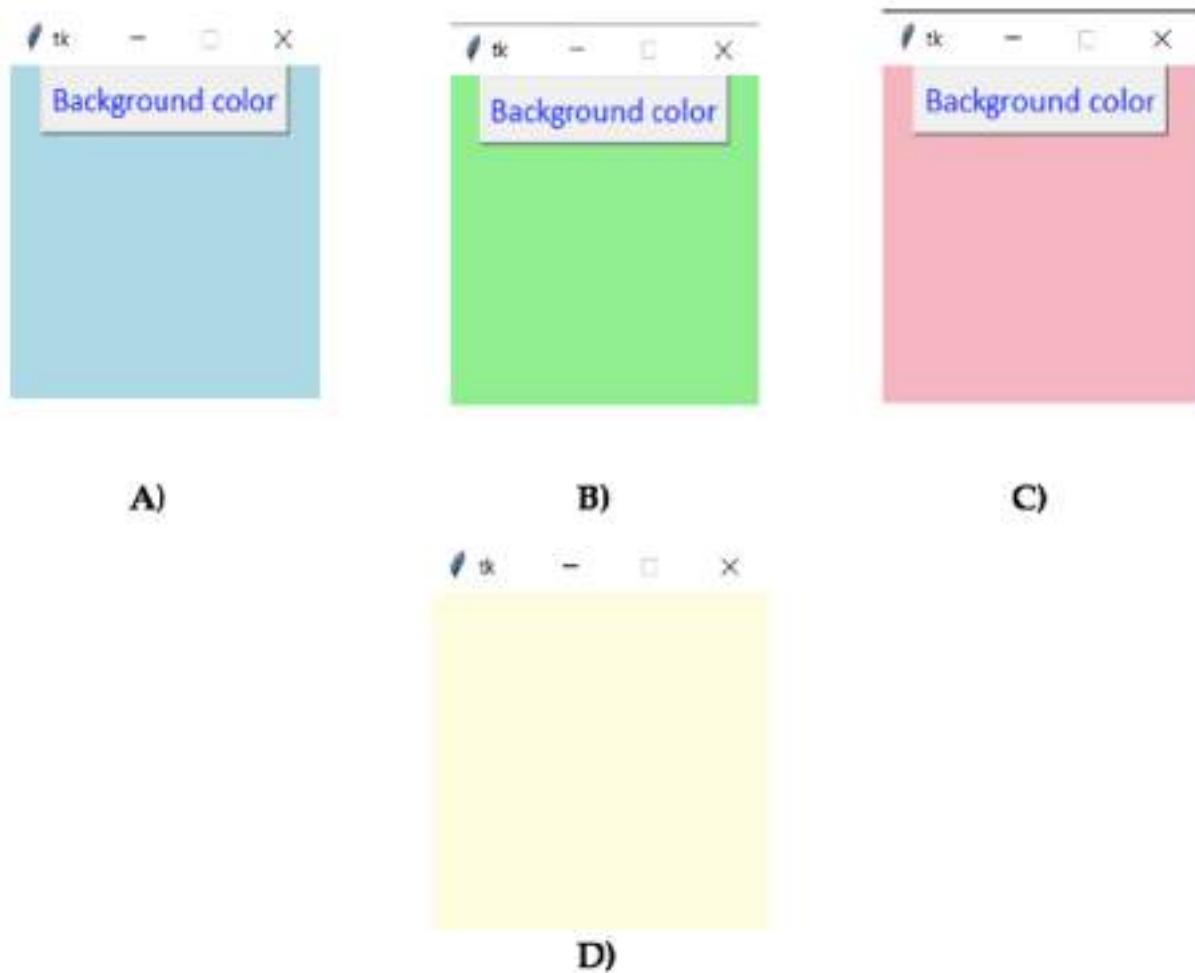


Figure 3.21: Output

Here,

- A) On pressing either *Shift*, *Alt*, or *Ctrl* and Up keys, the window color changes to LightBlue.
- B) On pressing either *Shift*, *Alt*, or *Ctrl* and Down key, the window color changes to LightGreen.
- C) On pressing either *Shift*, *Alt*, or *Ctrl* and Left key, the window color changes to LightPink.

D) On pressing either *Shift*, *Alt*, or *Ctrl* and Right key, the window color changes to LightYellow.

Note: The preceding code is covered in Program Name: Chap3_Example15.py

We can also perform event handling by pressing and releasing a button:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

mybutton1 = Button(myroot, text = 'Click Me!!!', font = ('Ari-
al',12))
mybutton1.bind('<Button>',lambda e: myroot.configure(back-
ground = 'LightBlue')) # on Mouse pressing Button
mybutton1.bind('<ButtonRelease>',lambda e: myroot.configure(back-
ground = 'Red')) # on Mouse releasing Button
mybutton1.pack()

myroot.mainloop()
```

Output:

Refer to *Figure 3.22*:



Figure 3.22: Output

Here,

- A) Output when the button is pressed and hold.
- B) Output when a button press is released.

Note: The preceding code is covered in Program Name: Chap3_Example16.py

In the above code, if the button is pressed, then the window color will be LightBlue and when released it will be Red.

We can also generate the same output with a single function:

```

from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.geometry('200x200') # but can be resized to any pixel un-
til we are using root.resizeable
myroot.resizable(0,0) # window size is fixed. cannot be larg-
er or smaller.

num = 1
def mydisplay(e):
    global num
    num = num + 1
    if num%2 == 0:
        myroot.configure(background = 'LightBlue')
    else:
        myroot.configure(background = 'Red')

mybutton1 = Button(myroot, text = 'Click Me!!!', font = ('Ari-
al',12))
mybutton1.bind('<Button>',mydisplay) # on Mouse pressing Button
mybutton1.bind('<ButtonRelease>',mydisplay) # on Mouse releas-
ing Button
mybutton1.pack()

myroot.mainloop()

```

Note: The preceding code is covered in Program Name: Chap3_Example17.py

In the above code, we have used a single function and global variable concept to change the window background color.

tkinter Checkbutton widget

This widget will allow the user to select more than one option by clicking the button corresponding to each option. So, multiple options can be selected

by the user at a time. A Yes/No choice is made by checking/unchecking the menu.

The syntax is:

```
mychk1= Checkbutton(myroot , options...)
```

Here,

myroot is the parent window.

Some of the lists of options that can be used as key-value pairs and are separated by commas are activeforeground, activebackground, background, bd, bitmap, cursor, command, disableforeground, fg, font, height, image, highlightcolor, justify, onvalue, offvalue, padx, pady, selectcolor, selectimage, text, state, underline, variable, width, and wraplength.

We have seen most of the options but some undiscussed options are as follows:

- **command:** This option will associate with a function whenever the **checkbutton** state is changed by the user.
- **onvalue:** This option will associate checkbutton's control variable default value to 1 when it is on or set. By setting onvalue to that value, an alternate value can be supplied for the on state.
- **offvalue:** This option will associate checkbutton's control variable default value to 0 when it is off or cleared. By setting offvalue to that value, an alternate value can be supplied for the off state.
- **text:** This option will display the text next to the checkbutton. Multiple lines of text can be displayed using '\n'.
- **variable:** This option will track the checkbox current state. It is an **IntVar** variable where 0 means off and 1 means set and will toggle between offvalue and onvalue when the button widget is pressed.

Refer to the following code:

```
from tkinter import *
from tkinter.ttk import *

myroot = Tk()
myroot.geometry('300x150')
myroot.title('CheckButton widget')

def myget():
    if i2.get() == 'check':
        s1.set('Checked')
    else:
        s1.set('UnChecked')

i2 = StringVar()
myc2 = Checkbutton(myroot, text = 'Check/Uncheck', variable = i2, offvalue = 'uncheck', onvalue = 'check', command = myget)
myc2.pack()

s1 = StringVar()
mye1 = Entry(myroot, font = ('Calibri',12), textvariable= s1)
mye1.pack(pady = 10)

myroot.mainloop()
```

Output:

Refer to *Figure 3.23*:

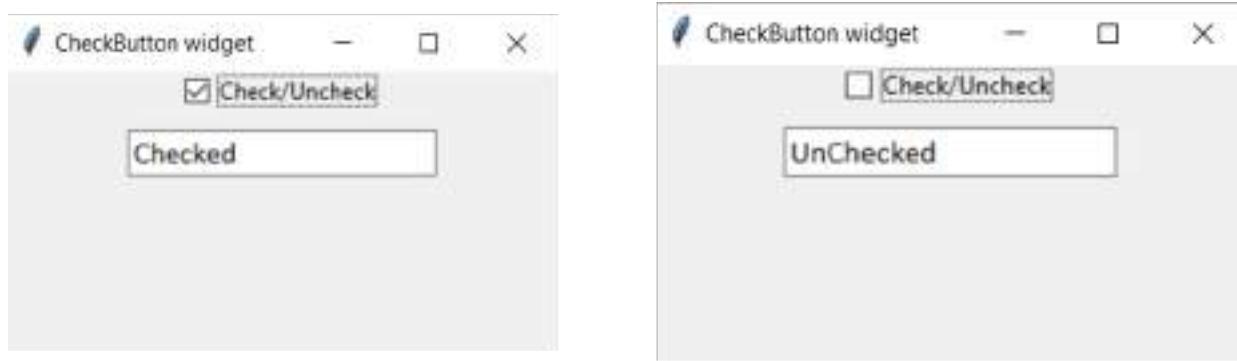


Figure 3.23: Output

Note: The preceding code is covered in Program Name: Chap3_Example18.py

In the next code, when the user clicks the **CheckButton**, the text checked will be written in the Entry widget as its value for on state is checked. When the user unchecks the **CheckButton**, the text **UnChecked** will be written in the Entry widget as its value for the off state is unchecked.

selectcolor: This option will set the checkbutton color when the widget is selected. The default is ‘Red’ color. If the indicator is True, the color is applied to the indicator. In Windows irrespective of the select state, it is used as the background for the indicator. When the indicator is set to False, the color is used as the background color for the entire widget whenever it is selected.

Refer to the following code:

```
from tkinter import *
myroot = Tk()
def selectcolor_indicatoronTrue():
    mychk1['selectcolor'] = 'Green'

def selectcolor_indicatoronFalse():
    mychk2['selectcolor'] = 'Blue'

mychk1 = Checkbutton(myroot, text = 'CheckButton', command = selectcolor_indicatoronTrue, indicatoron = True)
mychk1.place(x = 50, y = 50)
mychk2 = Checkbutton(myroot, text = 'CheckButton', command = selectcolor_indicatoronFalse, indicatoron = False)
mychk2.place(x = 50, y = 100)
myroot.mainloop()
```

Output:

Refer to *Figure 3.24*:

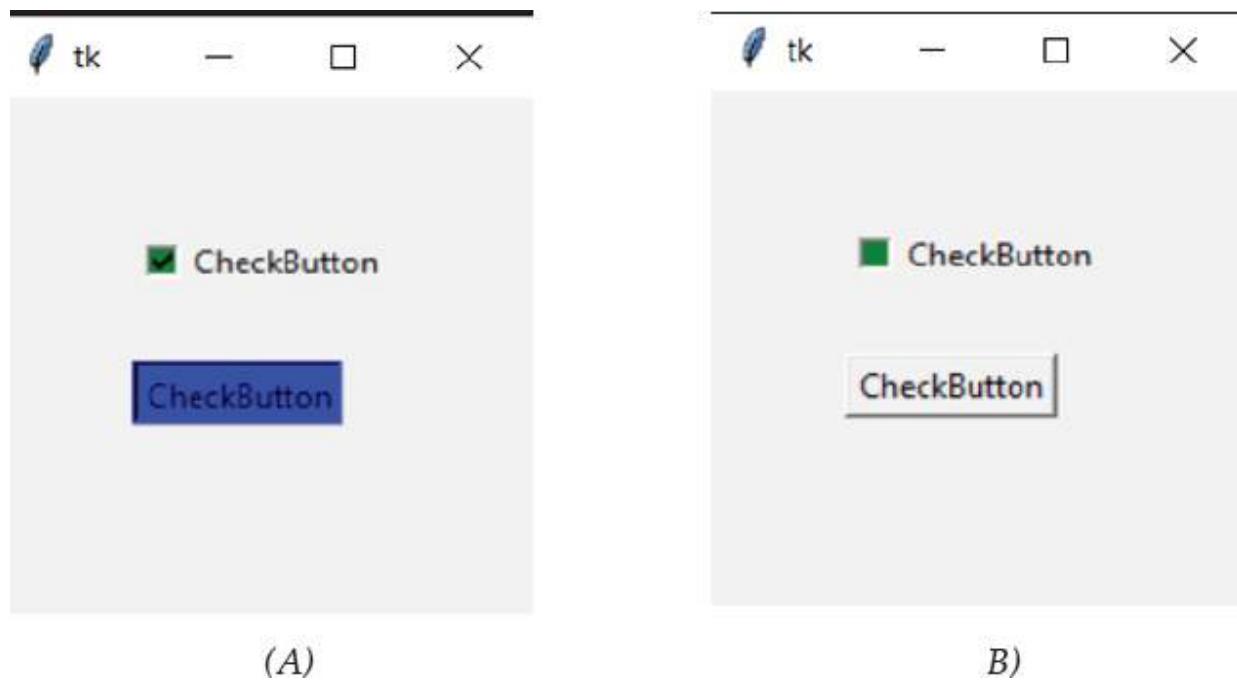


Figure 3.24: Output

Here,

- A) When both the Checkbuttttons are clicked.
- B) When both the Checkbuttttons are deselected.

Note: The preceding code is covered in Program Name: Chap3_Example19.py

image: This option will allow you to get the image displayed in the widget.

selectimage: This option will allow setting the image to the checkbutton.

Refer to the following code:

```
from tkinter import *
myroot = Tk()

myon_image = PhotoImage(width=50, height=25)
myoff_image = PhotoImage(width=50, height=25)
myon_image.put(("Light-
Green",), to=(0, 0, 24,24)) # It will put row formatted col-
ors to image starting from position T0
myoff_image.put(("Red",), to=(25, 0, 49, 24))

myval1 = IntVar(value=0)
myval2 = IntVar(value=1)
cb1 = Checkbutton(myroot, image=myoff_image, selectimage=myon_im-
age, indicatoron=False,
                    onvalue=1, offvalue=0, variable=myval1)
cb2 = Checkbutton(myroot, image=myoff_image, selectimage=myon_im-
age, indicatoron=False,
                    onvalue=1, offvalue=0, variable=myval2)

cb1.pack(padx=10, pady=10)
cb2.pack(padx=10, pady=10)

myroot.mainloop()
```

Output:

Refer to *Figure 3.25*:

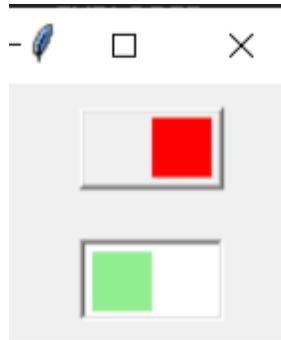


Figure 3.25: Default output of Chap3_Example20.py

In this code, we are setting the image option for the unselected state and the selectimage option for the selected state. There is another option called indicator on which we have set to False for not displaying the default indicator by the tkinter.

We can change the status of each check button as shown in *Figure 3.26*:

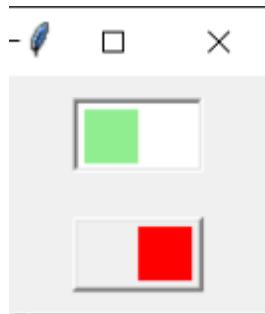


Figure 3.26: Output of Chap3_Example20.py on status change

Note: The preceding code is covered in Program Name: Chap3_Example20.py

state: This option when set to DISABLED, will make the widget unresponsive. The default state is NORMAL.

Refer to the following code:

```
from tkinter import *
myroot = Tk()
myroot.geometry('300x300')
def myselected():
    mychk1.config(state=NORMAL)

def mydisabled():
    mychk1.config(state=DISABLED)

mybtn1 = Button(myroot, text = 'Normal', command = myselected)
mybtn1.place(x = 50, y = 50)
mybtn2 = Button(myroot, text = 'Disabled', command = mydisabled)
mybtn2.place(x = 50, y = 100)

mychk1 = Checkbutton(myroot, text = 'CheckButton')
mychk1.place(x = 100, y = 150)

myroot.mainloop()
```

Refer to *Figure 3.27*:

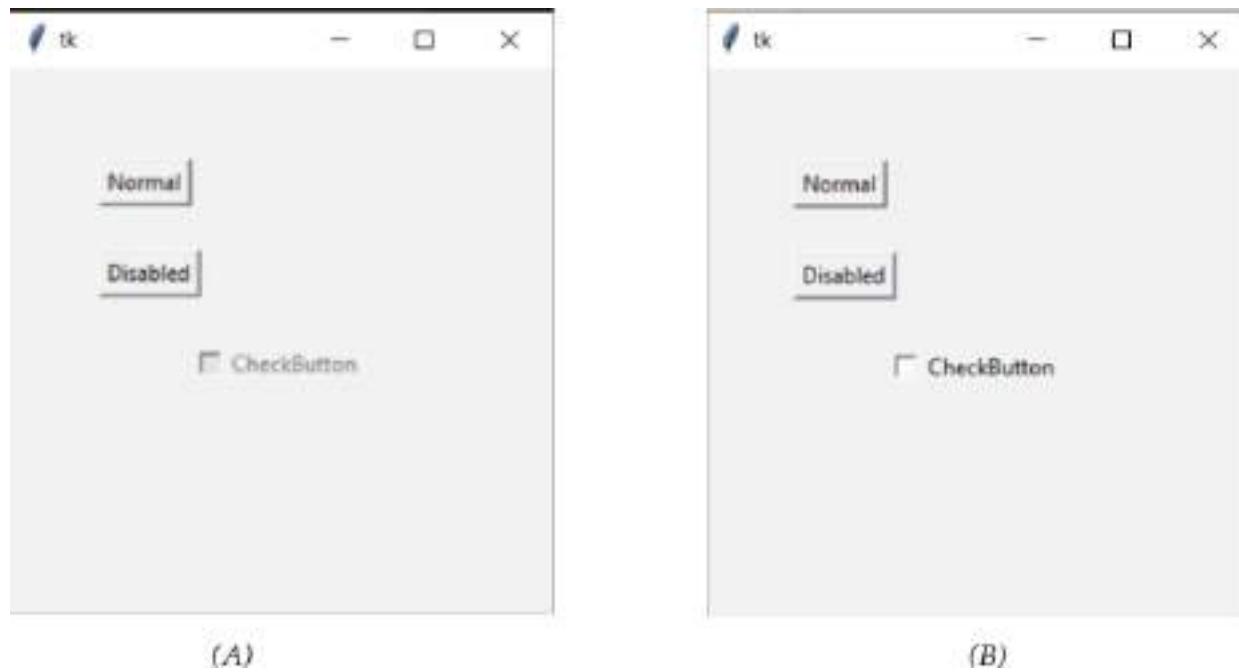


Figure 3.27: Output

Here,

- A) Output when the **Disabled** button is clicked
- B) Output when the **Normal** button is clicked

Note: The preceding code is covered in Program Name: Chap3_Example21.py

In the given code, we can see that when the state of the checkbutton is DISABLED, the checkbutton becomes unresponsive. It is brought back to state = NORMAL, when the Normal button is clicked.

We shall see an example of a **CheckButton** where we are using its maximum options:

```
from tkinter import *

myroot = Tk()
myroot.geometry('300x150')
myroot.title('CheckButton widget')

mynum1 = IntVar()
mynum2 = IntVar()
mys1 = StringVar()

def mydatainsertion():

    if mynum1.get() == 1 and mynum2.get() == 0:# reading status of
checkbutton
        mys1.set("Python")# setting the value to the Entry widget

    if mynum1.get() == 0 and mynum2.get() == 1:
        mys1.set("C#.Net")

    if mynum1.get() == 1 and mynum2.get() == 1:
        mys1.set("I love to study both")

    if mynum1.get() == 0 and mynum2.get() == 0:
        mys1.set("I hate to study both")

myc1 = Checkbutton(myroot, variable = mynum1, font = ('Cal-
ibri',12), text = 'Python', command = mydatainsertion)
```

```
myc1.pack()

myc2 = Checkbutton(myroot,variable = mynum2, font = ('Calibri',12), text = 'C#.Net', command = mydatainsertion)
myc2.pack()

mye1 = Entry(myroot, font = ('Calibri',15), textvariable = mys1)
mye1.pack()

myroot.mainloop()
```

Output:

Refer to *Figure 3.28*:

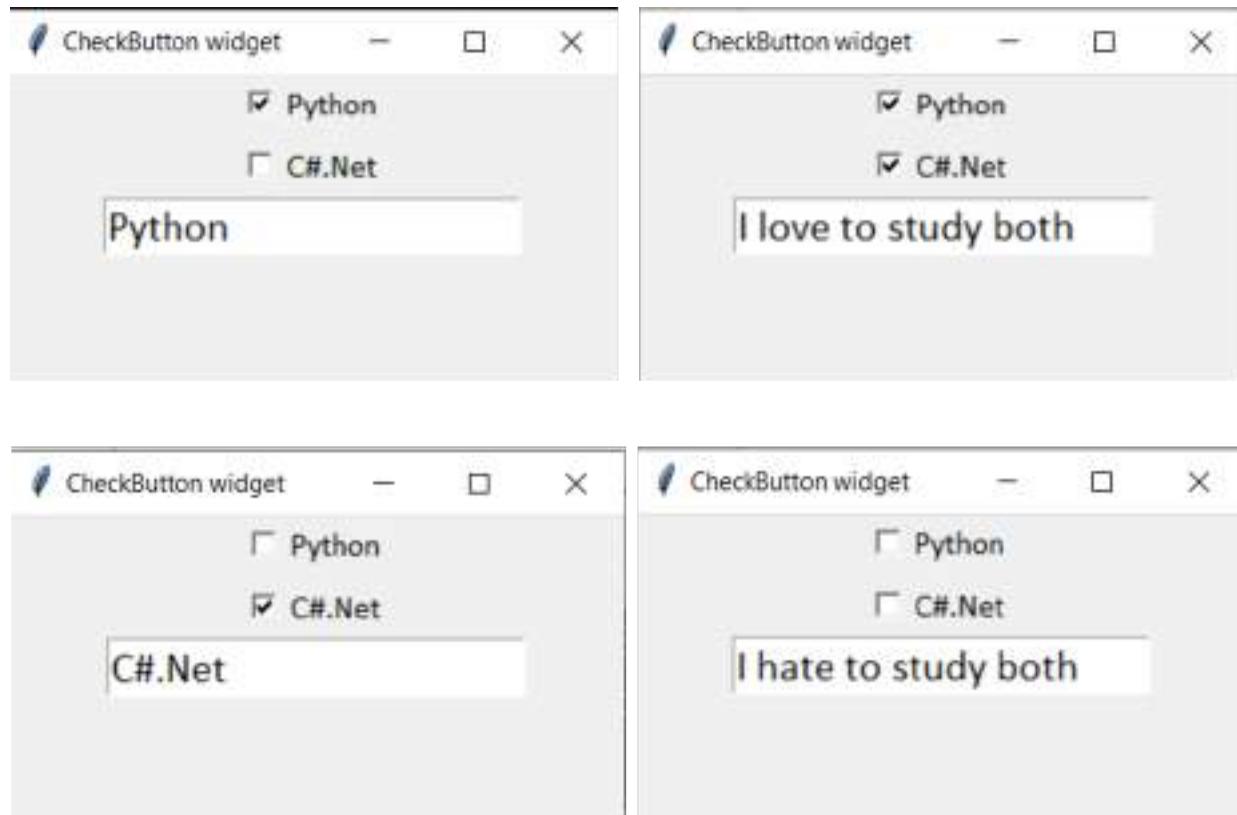


Figure 3.28: Output

Note: The preceding code is covered in Program Name: Chap3_Example22.py

In the given code, we have linked the variables **mynum1** and **mynum2** to the **checkbutton**, as these values will be used for reading and writing the status of the checkbuttons. We are reading the status of the checkbuttons (by the value associated with the variables) and set the value of the checkbuttons.

Some of the commonly used methods associated with this widget are:

- **select**: This method will set the value to onvalue as it will set the checkbutton.
- **deselect**: This method will set the value to offvalue as it will deselect the checkbutton.
- **flash**: This method will allow checkbutton to flash between normal and active colors.
- **invoke**: This method will call the command if the checkbutton is clicked by the user for changing the state.
- **toggle**: This method will toggle between different checkbuttons.

Let us see an example for a better understanding of these methods:

```

from tkinter import *
myroot = Tk()
myroot.geometry('300x250')
def myselected():
    mychk1.select()

def mydeselect():
    mychk1.deselect()

def mytoggle():
    mychk1.toggle()

def myinvoke():
    myl1 = Label(myroot, text = 'CheckStat')
    myl1.place(x = 20, y = 150)

mybtn1 = Button(myroot, text = 'Select', command = myselected)
mybtn1.place(x = 50, y = 50)
mybtn2 = Button(myroot, text = 'Deselect', command = mydeselect)
mybtn2.place(x = 50, y = 100)
mybtn3 = Button(myroot, text = 'Toggle', command = mytoggle)
mybtn3.place(x = 150, y = 50)
mybtn4 = Button(myroot, text = 'Invoke', com-
mand = myinvoke) # will call the command associated with the but-
ton initially on run
mybtn4.place(x = 150, y = 100)
mybtn4.invoke()

mychk1 = Checkbutton(myroot, text = 'CheckButton')
mychk1.place(x = 100, y = 150)

myroot.mainloop()

```

Output:

Refer to *Figure 3.29*:

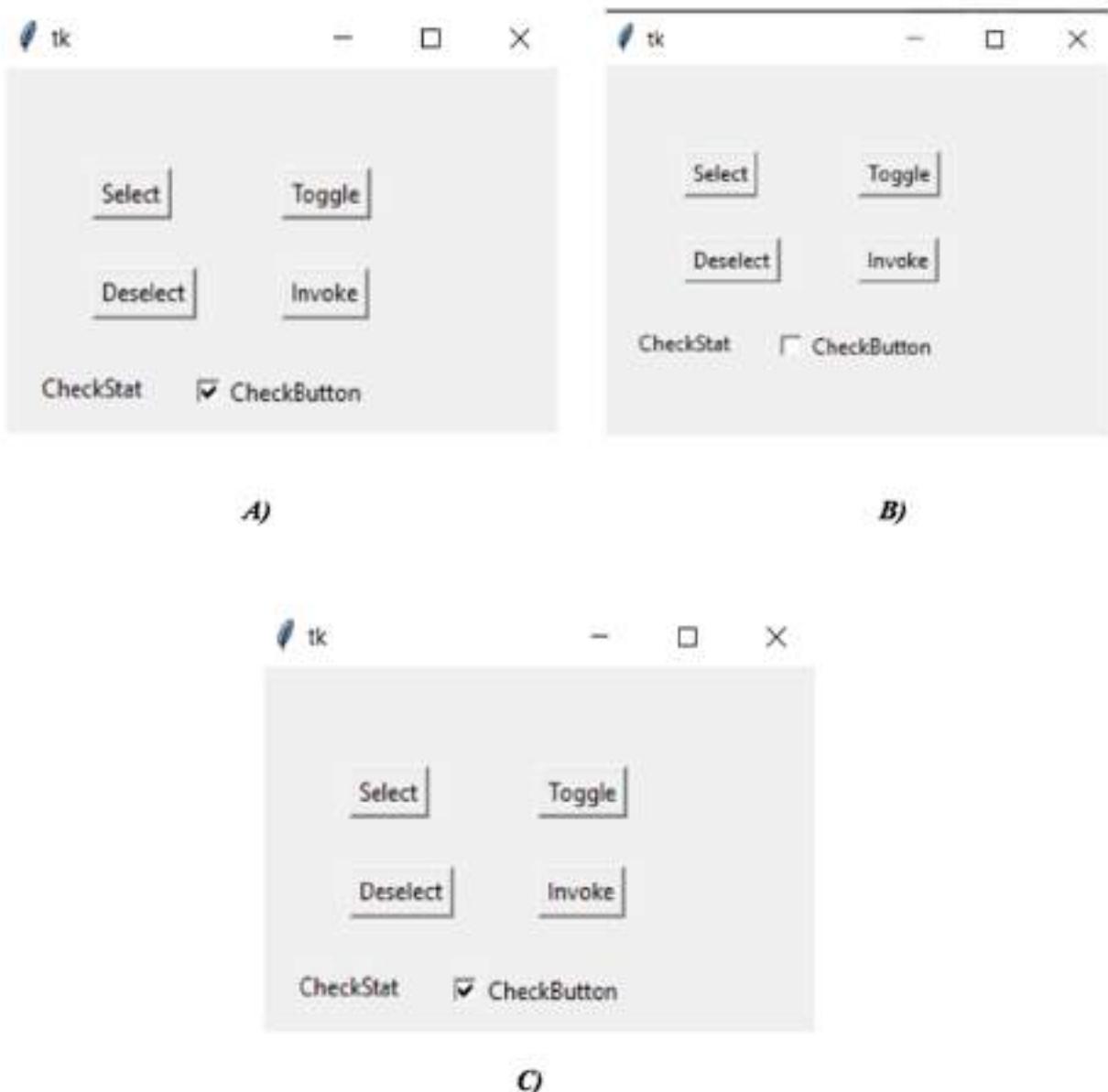


Figure 3.29: Output

Here,

- A) When the Select Button is clicked
- B) When Deselect Button is clicked.
- C) When the value is getting toggled by clicking the Toggle button.

Note: The preceding code is covered in Program Name: Chap3_Example23.py

We can see that on clicking the Select/Deselect button, the state of the checkbox is modified by using the select/deselect method and is toggled by using the toggle method.

tkinter Radiobutton widget

This widget will give the user multiple options where the user can select any one of the options. Multiple line text or images can be displayed on the radiobuttons.

The syntax is:

```
myr1= Radiobutton(myroot, options...)
```

where,

myroot is the parent window. Some of the lists of options that can be used as key-value pairs and are separated by commas are anchor, activebackground, activeforeground, bg, bitmap, command, bd, font, cursor, height, fg, highlightbackground, image, selectimage, highlightcolor, justify, padx, pady, selectcolor, state, text, textvariable, value, relief, underline, variable, width, and wrapline.

We have seen most of the options but some undiscussed options are as follows:

- **command:** This option will be associated with a function whenever the radiobutton state is changed by the user.
- **value:** Each radiobutton value will be assigned to the control variable when it is turned on by the user. Each radiobutton in the group will give a different integer value option, when the control variable is an IntVar. Each radiobutton in the group will give a different string value option, when the control variable is an StringVar.
- **image:** This option will allow getting the image displayed in the widget instead of the text.

- **selectimage:** This option will display the image on the radiobutton when it is selected.
- **selectcolor:** This option will set the radiobutton color when selected. The default color is Red.
- **state:** This option when set to DISABLED, will make the widget unresponsive. The default state is NORMAL.
- **text:** This option will display the text next to the radiobutton. Multiple lines of text can be displayed using '\n'.
- **textvariable:** This option allows us to update the message text, whenever we want and it is of String type.
- **variable:** This option displays the control variable which keeps track of the user choices and monitors radiobutton state.
- **indicatoron:** This option can allow the radiobuttons with complete text in a box when setting the indicator on the option to 0.

Let us see the usage of these options with some examples:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('200x200')

COLOR1 = 'LightGreen'
COLOR2 = 'LightBlue'

def mydisplay():
    if myi1.get() == 1:
        myroot.configure(bg = COLOR1)
    elif myi1.get() == 2:
        myroot.configure(bg = COLOR2)

myi1 = IntVar()
myr1 = Radiobutton(myroot, text = COLOR1, value = 1, variable = myi1)
myr1.pack()

myr2 = Radiobutton(myroot, text = COLOR2, value = 2, variable = myi1)
myr2.pack()

mybtn = Button(myroot, text = 'Background_Click', command = mydisplay)
mybtn.pack()

myroot.mainloop() # display window until we press the close button
```

Output:

Refer to *Figure 3.30*:

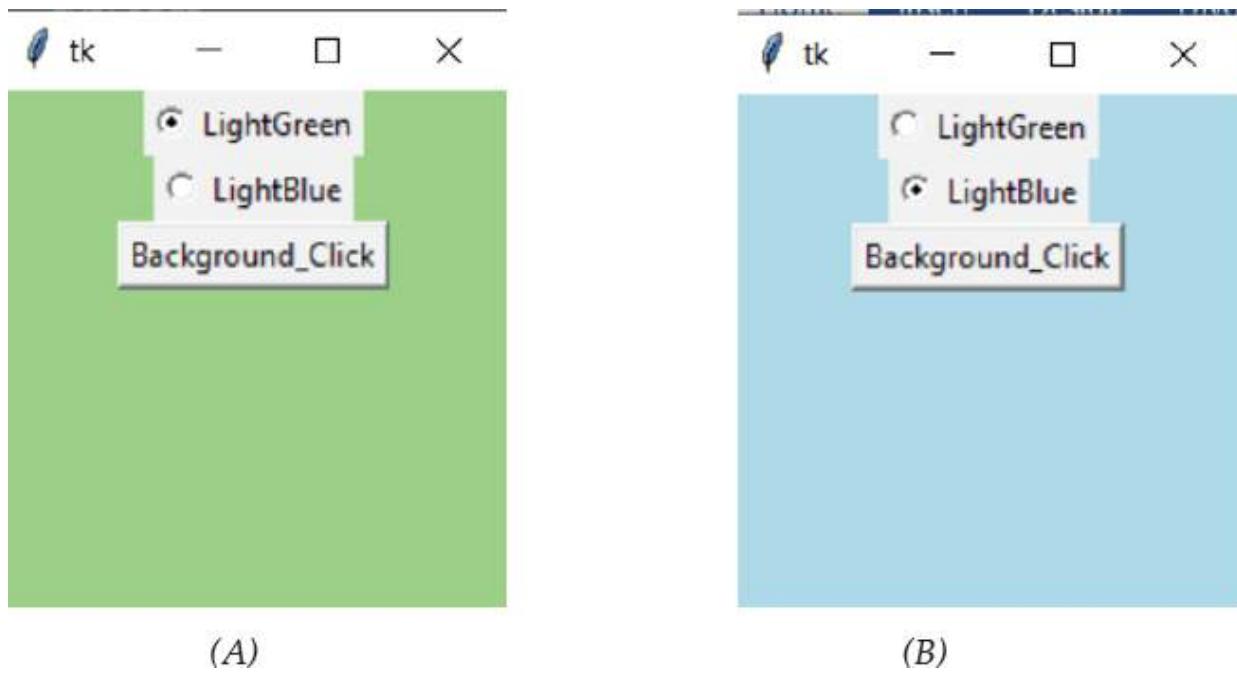


Figure 3.30: Output

Here,

- A) When RadiouButton1 LightGreen is selected and the Button is clicked.
- B) When RadiouButton2 LightBlue is selected and the Button is clicked.

Note: The preceding code is covered in Program Name: Chap3_Example24.py

In this code, we are assigning the color names to the global variables (COLOR1, COLOR2). A callback function **mydisplay** will change the background color of the main form, based on the user selection. We have created an **IntVar** variable. Only a single variable is created to be used by all 2 radiobuttons. If we select any radiobutton, then the other radiobutton will be unselected. We have created 2 radiobuttons and assigned them to the main form. The variable is then passed to be used in the callback function which will create the changing of background color of the window. So, when LightGreen radiobutton is selected, then the background color of the main form will be LightGreen. When LightBlue radiobutton is selected, then the background color of the main form will be LightBlue.

We can display the same output by using the command option when we are creating radiobutton and removing the button widget, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('200x200')

COLOR1 = 'LightGreen'
COLOR2 = 'LightBlue'

def mydisplay():
    if myi1.get() == 1:
        myroot.configure(bg = COLOR1)
    elif myi1.get() == 2:
        myroot.configure(bg = COLOR2)

myi1 = IntVar()
myr1 = Radiobutton(myroot, text = COLOR1, value = 1, variable = myi1, command = mydisplay)

myr2 = Radiobutton(myroot, text = COLOR2, value = 2, variable = myi1, command = mydisplay)
myr2.pack()

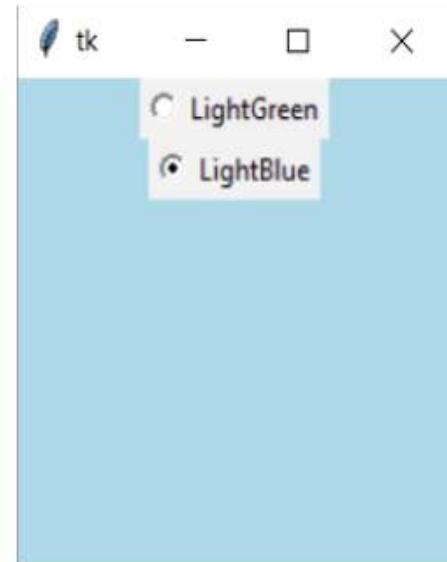
myroot.mainloop() # display window until we press the close button
```

Output:

Refer to *Figure 3.31*:



A)



B)

Figure 3.31: Output

Here,

- A) When RadiouButton1 LightGreen is selected.
- B) When RadiouButton2 LightBlue is selected.

Note: The preceding code is covered in Program Name: Chap3_Example25.py

We can display the image to a radiobutton as shown:

```
from tkinter import *
myroot = Tk()

myon_image = PhotoImage(width=50, height=25)
myoff_image = PhotoImage(width=50, height=25)
myon_image.put(("Light-Green",), to=(0, 0, 24,24)) # It will put row formatted colors to image starting from position T0
myoff_image.put(("Red",), to=(0, 0, 24, 24))

myrbvar = IntVar(value=1)
myrb1 = Radiobutton(myroot, variable=myrbvar, value=0, bd=0,
                     text="RadioButton1", compound="left", indicator=False,
                     image=myoff_image, selectimage=myon_image)
myrb2 = Radiobutton(myroot, variable=myrbvar, value=1, bd=0,
                     text="RadioButton2", compound="left", indicator=False,
                     image=myoff_image, selectimage=myon_image)

myrb1.pack(padx=10, pady=10)
myrb2.pack(padx=10, pady=10)

myroot.mainloop()
```

Output:

Refer to *Figure 3.32*:

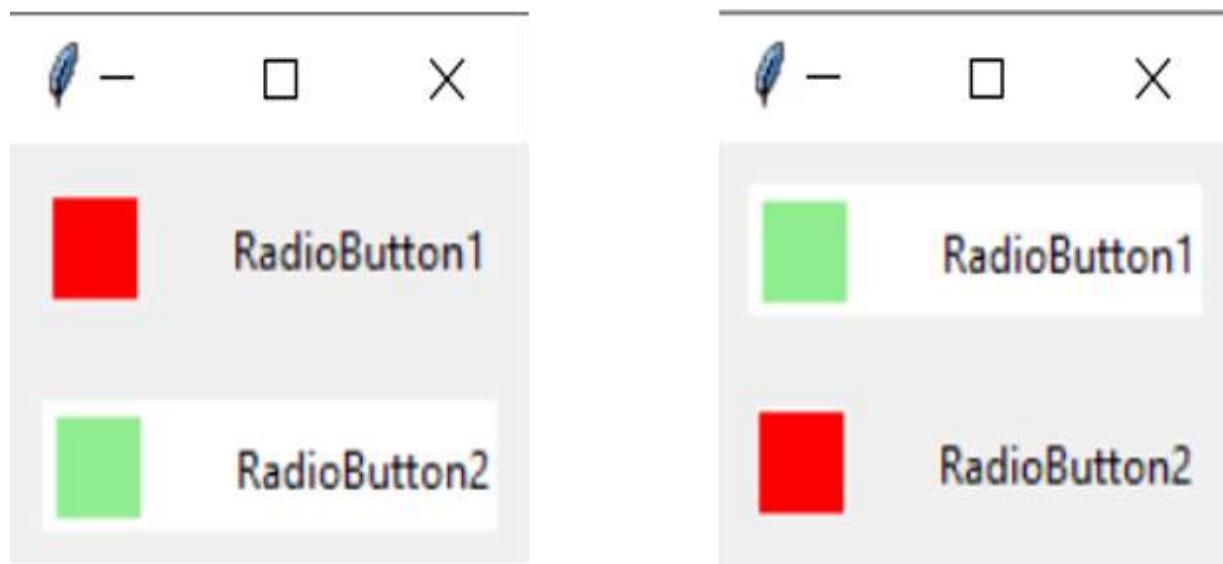


Figure 3.32: Output

Note: The preceding code is covered in Program Name: Chap3_Example26.py

In the given code, we can see that the image is getting displayed in the radiobutton. Our images have been used for the selectors with the radiobutton attributes, image and selectimage, which is used in conjunction with compound, borderwidth and indicatoron.

We can display radiobutton with complete text in the box when indicatoron is set to 0. Refer to the following code:

```
from tkinter import *
myroot = Tk()

myrb1 = Radiobutton(myroot, value=0, text="RadioButton1", bg = 'lightGreen', indicatoron=False,)
myrb2 = Radiobutton(myroot, value=1, text="RadioButton2",bg = 'lightGreen', indicatoron=False)

myrb1.pack(padx=10, pady=10)
myrb2.pack(padx=10, pady=10)

myroot.mainloop()
```

Output:

Refer to *Figure 3.33*:

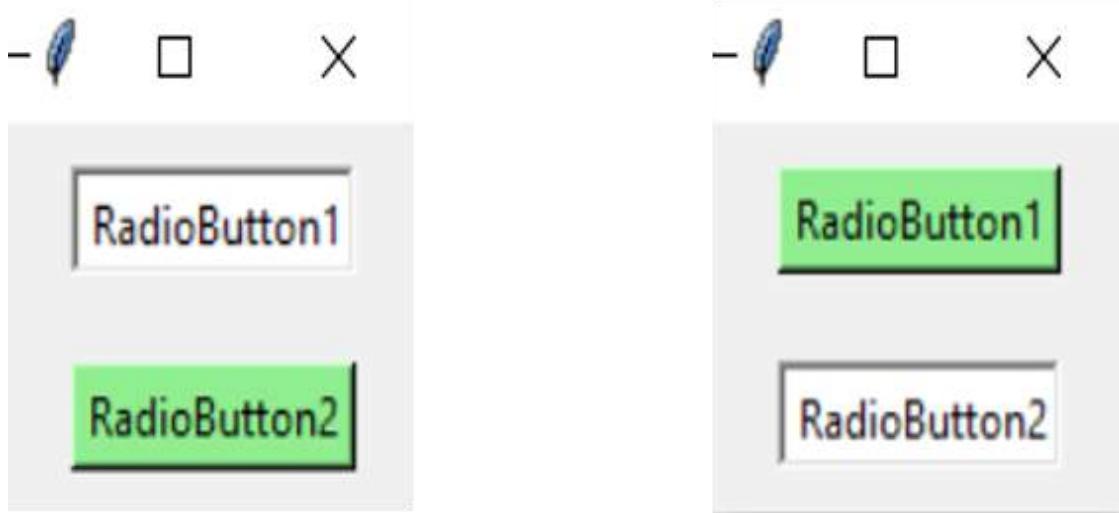


Figure 3.33: Output of Chap3_Example27.py

Note: The preceding code is covered in Program Name: Chap3_Example27.py

We can see that we have set the background of these radiobutton boxes as LightGreen, and the selected ones are sunken and are having a white background.

We can also set the color of the radiobutton when selected, as shown:

```
from tkinter import *
myroot = Tk()
def selectcolor_indicatoronTrue():
    mychk1['selectcolor'] = 'LightGreen'

def selectcolor_indicatoronFalse():
    mychk2['selectcolor'] = 'Blue'

mychk1 = Radiobutton(myroot, text = 'RadioButton1', com-
mand = selectcolor_indicatoronTrue, indicatoron = True, value = 1)
mychk1.place(x = 50, y = 50)
mychk2 = Radiobutton(myroot, text = 'RadioButton2', com-
mand = selectcolor_indicatoronFalse, indicatoron = False, value = 2)
mychk2.place(x = 50, y = 100)
myroot.mainloop()
```

Output:

Refer to *Figure 3.34*:

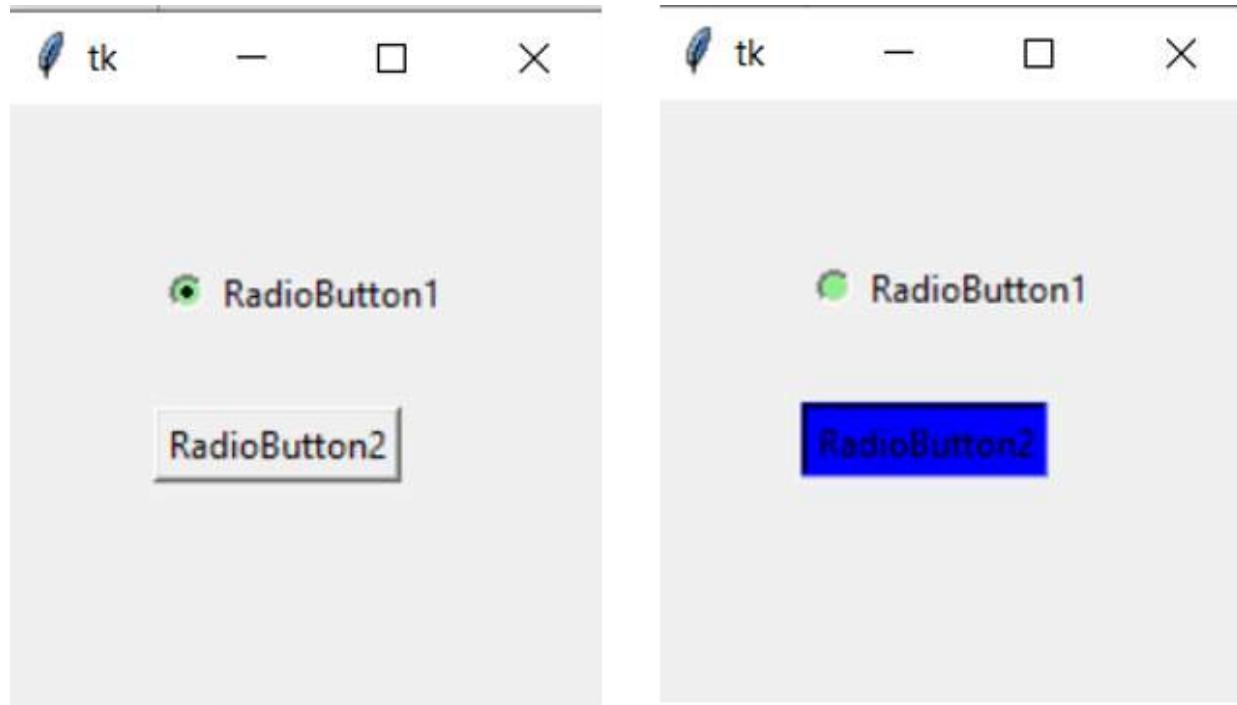


Figure 3.34: Output

Note: The preceding code is covered in Program Name: Chap3_Example28.py

In this code, we can see that when Radiobutton1 is selected, then its color is changed to LightGreen and when RadioButton2 is selected, then its color is changed to Blue.

Some of the commonly used methods associated with this widget are:

- **select**: This method will set/select the radiobutton.
- **deselect**: This method will deselect/turn off the radiobutton.
- **flash**: This method will allow radiobutton to flash between normal and active colors several times.
- **invoke**: This method will call mandatory action when radiobutton state is changed.

We shall see some examples related to these methods:

```
from tkinter import *
myroot = Tk()

def myselect():
    mychk2.select()
    mychk2['selectcolor'] = 'LightGreen'

def mydeselect():
```

```
mychk2.deselect()
mychk2['bg'] = 'Red'

mychk1 = Radiobutton(myroot, text = 'RadioButton1', indicatoron = True, value = 2)
mychk1.place(x = 50)
mychk1.invoke()

mychk2 = Radiobutton(myroot, text = 'RadioButton2', indicatoron = False, value = 1)
mychk2.place(x = 50, y = 50)

mybtn1 = Button(myroot, text = 'Select', command = myselect)
mybtn1.place(x = 50, y = 100)

mybtn2 = Button(myroot, text = 'Deselect', command = mydeselect)
mybtn2.place(x = 50, y = 150)

myroot.mainloop()
```

Initial Output RadioButton1 is invoked:

Refer to *Figure 3.35*:



Figure 3.35: Output

Refer to *Figure 3.36*:

Output when **Select** button is clicked Output when **Deselect** button is clicked.

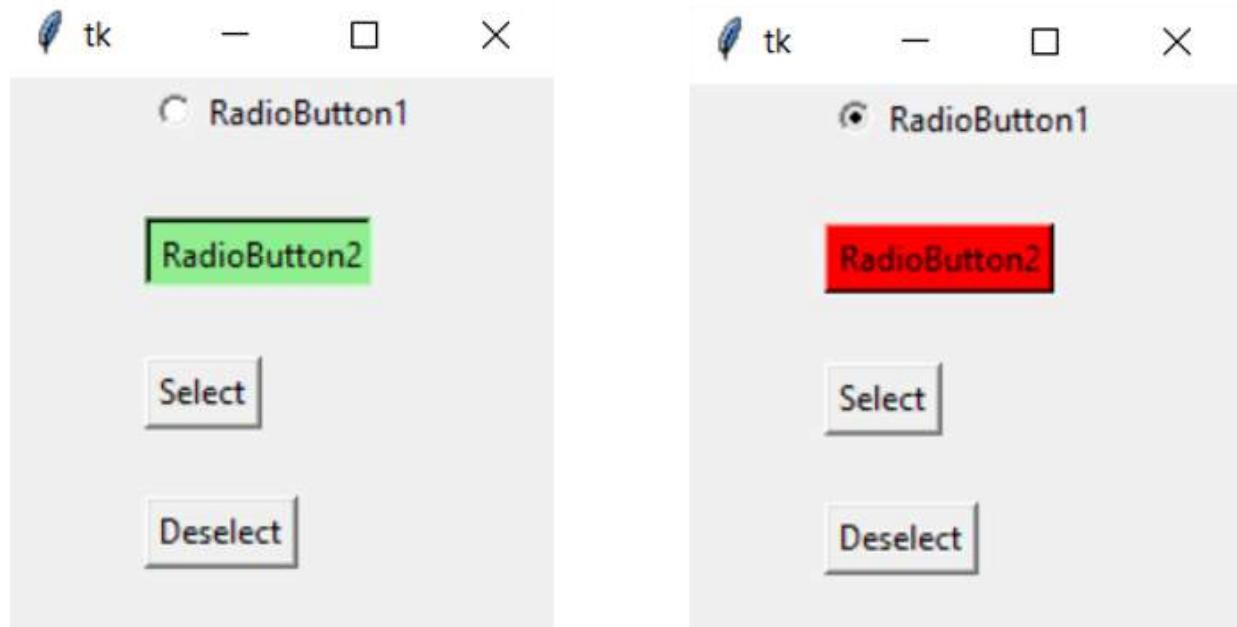


Figure 3.36: Output

Note: The preceding code is covered in Program Name: Chap3_Example29.py

In the above code, Radiobutton1 is activated by default when the GUI code is run. When the select button is clicked, then RadioButton2 is selected and RadioButton1 is deselected. When Deselect button is clicked, then RadioButton2 is deselected and RadioButton1 is selected as shown below.

tkinter OptionMenu widget

This widget creates a pop-menu and button where the user can select one option at a time from a list of options.

We have to pass in the **tkinter** variable to get the currently selected value from an options menu:

```
from tkinter import *

myroot = Tk()

myroot.title("Fruit Selection")
myroot.geometry('300x200')

# Tkinter variable is created
myvar = StringVar()
myvar.set("Litchi")
```

```
# Create an option menu by passing the variable and option list
myselection = OptionMenu(myroot, myvar, "Mango", "Apple", "Li-
tchi", "Banana") # variable bound to option menu
myselection.pack()

# Create button with command
def mydisplay():
    print("The chosen value :", myvar.get())

mybtn_show = Button(myroot, text="Myshow", command=mydisplay)
mybtn_show.pack(pady = 30, side = LEFT, anchor = N)

myroot.mainloop()
```

Output:

Refer to *Figure 3.37*:

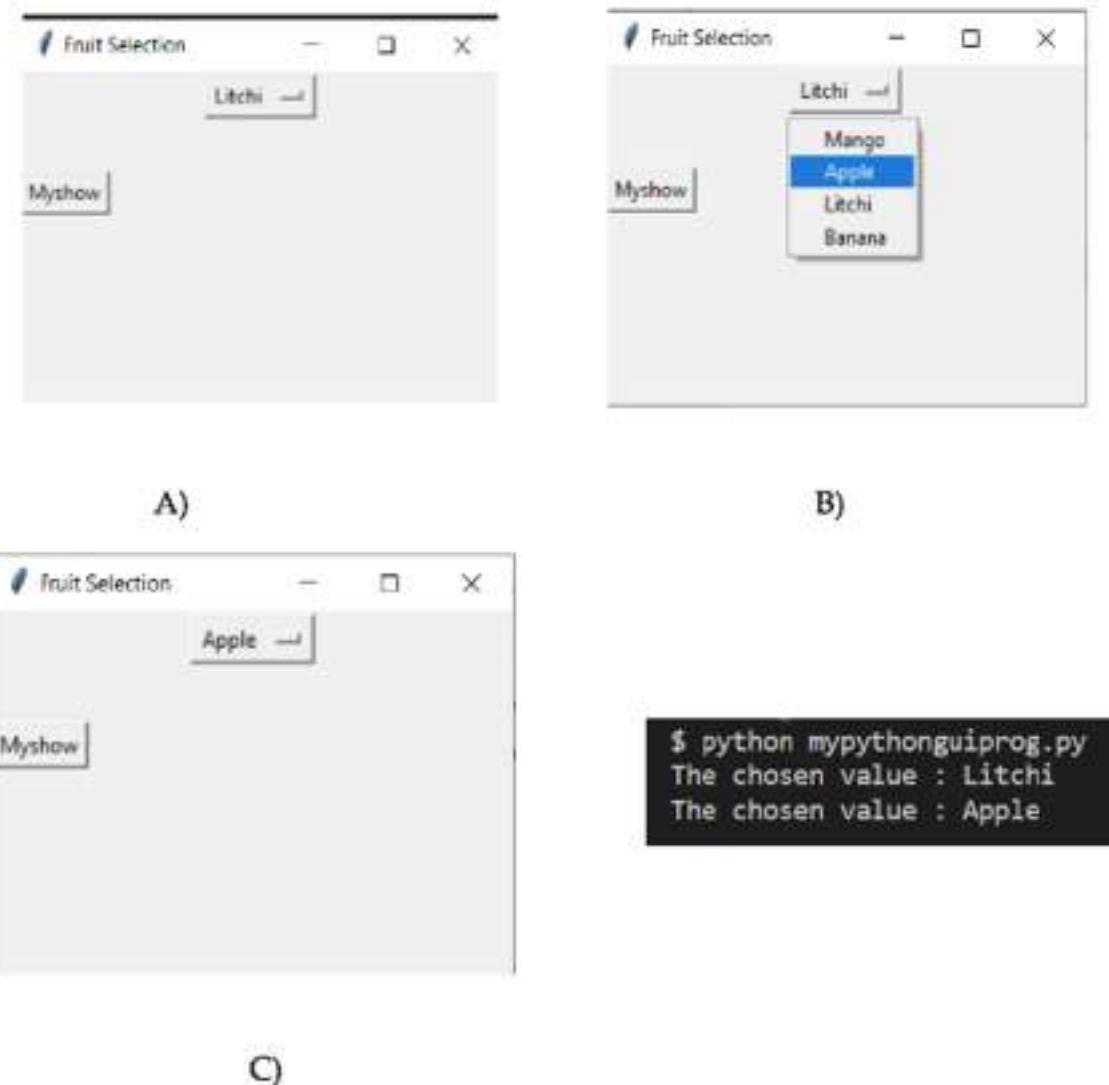


Figure 3.37: Output of *Chap3_Example30.py*

Note: The preceding code is covered in Program Name: **Chap3_Example30.py**

In this code, a `tkinter` variable is created, which is bound to the `OptionMenu` which will read the currently selected option from `OptionMenu` and will set the current value for the menu. Here, ‘Litchi’ is set as the current value (A). An `OptionMenu` widget is created with the first parameter as the parent widget and the remaining parameters as options. Users can select any value by clicking the button such that a pop menu will be shown (B). We have selected Apple and the text display is shown on the button (C). A button ‘Myshow’ is created with a command such that whenever the user clicks on

it, the selected value is taken from the OptionMenu and displayed on the console.

We can generate the same output by creating OptionMenu from the option list as shown here:

```
from tkinter import *

myroot = Tk()

myroot.title("Fruit Selection")
myroot.geometry('300x150')

# List is created
myoptions = ['Litchi', "Mango", "Apple", "Banana"]

# Tkinter variable is created
myvar = StringVar(myroot)
myvar.set(myoptions[0])

# Create an option menu by passing the variable and option list
myselection = OptionMenu(myroot, myvar, *myoptions) # variable bound to option menu
myselection.pack()

# Create button with command
def mydisplay():
    print("The chosen value :", myvar.get())

mybtn_show = Button(myroot, text="Myshow", command=mydisplay)
mybtn_show.pack(pady = 30, side = LEFT, anchor = N)

myroot.mainloop()
```

Note: The preceding code is covered in Program Name: Chap3_Example31.py

Conclusion

In this chapter, we learned about the four important tkinter widgets where we saw their most commonly used options with multiple examples. We saw multiple user interactive examples highlighting their applications. We saw how an event was binded with a function with their important arguments. The binding of events to these widgets with multiple examples and different methods including lambda expressions was explained with examples. The instance level, application level and class level binding were well explored. Moreover, the background color of the main window was changed based on mouse enter, mouse leave, function press, keypress events and so on. Most importantly we need to know when to use these four widgets as per our need.

Points of remember

- Button widget in tkinter is the most commonly used widget for creating GUI applications in tkinter.
- Checkbutton widget gives the provision to the user to select more than one option.
- Radiobutton widget gives the provision to the user to select exactly one of the predefined sets of options will be chosen.
- Option-Menu widget allows the user to display how a pop-menu and button widget will be created for an individual option selection from a list of options.
- Binding an event to a widget instance is called instance binding. Binding an event to entire application is called application-level binding. Binding an event to a particular class level is called class-level binding.

Questions

1. Explain the usage of the tkinter Button widget.
2. Which widget is used for interaction with the user, explain in detail.

3. Write short notes on Events and Bindings.
4. Explain different event types in Tkinter.
5. Explain the usage of the tkinter Check button widget.
6. How to select multiple options, and explain the widget used for this purpose.
7. Explain the usage of the tkinter Radiobutton widget.
8. Explain the usage of the tkinter OptionMenu widget.
9. Which widget is used to create a pop-menu and button where the user can select one option at a time from a list of options? Explain in detail with an example.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 4

Getting Insights of Input Widgets in tkinter

Introduction

In tkinter, users can enter data into a programme using input widgets, which are GUI components. There are several different types of input widgets, including text fields, check boxes, radio buttons, and drop-down menus. Using the suitable classes and methods offered by the library, these widgets can be added to a GUI window in tkinter. After being introduced, input widgets can be customized to meet the needs of the application by establishing default values, applying validation criteria, or imposing input volume restrictions. They can also be connected to programme logic, so that when a user interacts with them, a variable is updated or an event is started. Overall, input widgets offer a straightforward and understandable way to interact with software and enter data, and they are a crucial part of the majority of GUI programmes.

Structure

In this chapter, we will discuss the following topics:

- tkinter Entry widget
- tkinter Scrollbar widget
- tkinter Spinbox widget
- tkinter Scale widget
- tkinter Text widget

- tkinter Combobox widget

Objectives

In this chapter, the reader will firstly learn about the creation of a simple GUI app using the tkinter Entry widget in a very neat way with various options and explanations, followed by different examples. Then, the validation concept in the Entry widget is also explained. Next, we shall see about the scrollbar widget where the user will look into the scrolling capability in the vertical or horizontal direction with different widgets such as List Box, Entry, and Text. Another one is the tkinter Spinbox widget where the range of input values will be fed to the user, out of which the user can select one. Next, we will explore how to implement a graphical slider to any Python application program by using the tkinter Scale widget followed by the concept of the tkinter Text widget, where the user can insert multiple text fields. Finally, we will be dealing with the tkinter Combobox widget and its applications.

tkinter Entry widget

A widget that accepts single-line text strings from the user. A single line of text can be entered or displayed in this widget. It generally comes with a label because it is unclear what the user should type if we are not mentioning labels. The addition of more than one input field is allowed.

The syntax is:

```
myl1= Entry(myroot , options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, command, bd, cursor, exportselection, font, highlightcolor, justify, fg, relief, selectborderwidth,

`selectbackground`, `selectforeground`, `state`, `show`, `textvariable`, `xscrollcommand`, and `width`.

We have seen the maximum options and will discuss those options which we have not discussed:

- **command**: Every time, the operation needs to happen when the state of the `Entry` widget is changed.
- **exportselection**: Whenever the text is selected within an `Entry` widget and if `exportselection` is set to 0, the automatic export to the clipboard is restricted.
- **selectborderwidth**: This option will use border width around selected text and the default is 1 pixel.
- **state**: This option, when set to `DISABLED`, will make the `Entry` widget unresponsive and will go out the control. The default state is `NORMAL`.
- **show**: When there is a requirement to display some sort of special character in the `Entry` widget, this special character takes place in the actual text position. We all know that a password always takes a special character asterisk'*', whenever we try to log in to any account.
- **textvariable**: This option is set to `StringVar` class instance when there is a need to retrieve the current text from the `Entry` widget.
- **xscrollcommand**: This option will allow us to link the horizontal scrollbar to an `Entry` widget when we are entering more text than the actual widget width.

This example depicts the usage of `state` and `textvariable` options in the `Entry` widget, as shown:

```

from tkinter import *

class MySTATE:
    def __init__(self, myroot):
        self.myvar = StringVar()
        self.myvar.set('python')

        self.myl1 = Label(myroot, text = 'Normal state')
        self.myl1.grid(row = 0, column = 0)

        self.myl2 = Label(myroot, text = 'Disabled state')
        self.myl2.grid(row = 1, column = 0, pady = 10)

        self.myl3 = Label(myroot, text = ' Readonly state')
        self.myl3.grid(row = 2, column = 0, pady = 10)

        self.mye1 = Entry(myroot, textvariable=self.
myvar, width=15, state = 'normal')
        self.mye1.grid(row = 0, column = 1, padx = 10)
        self.mye2 = Entry(myroot, textvariable=self.
myvar, width=15, state = 'disabled')
        self.mye2.grid(row = 1, column = 1, padx = 10)
        self.mye3 = Entry(myroot, textvariable=self.
myvar, width=15, state = 'readonly')
        self.mye3.grid(row = 2, column = 1, padx = 10)

if __name__ == "__main__":
    myroot = Tk()
    myobj = MySTATE(myroot)
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 4.1*:

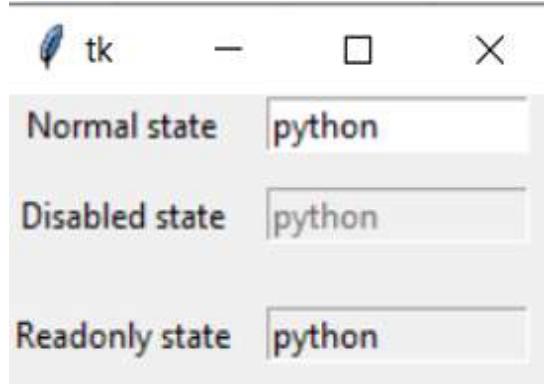


Figure 4.1: Output

Note: The preceding code is covered in Program Name: Chap4_Example1.py

In the above code, we are creating 3 **Entry** widgets in the parent widget having each state as normal, disabled, and read only. We can see that when the **Entry** widget is in the normal state, it can accept input from the user which can be changed if required. When the **Entry** widget is in a disabled or read only state, it means that it cannot be changed by the user.

Now, we shall see the usage of the **show** and **selectborderwidth** option using the entry widget, as shown:

```
from tkinter import *

class MyLogin:
    def __init__(self, myroot):
```

```

self.myl1 = Label(myroot, text = 'Username')
self.myl1.grid(row = 0, column = 0)

self.myl2 = Label(myroot, text = 'Password')
self.myl2.grid(row = 1, column = 0, pady = 10)

self.mye1 = Entry(myroot, width=15, selectborderwidth = 3)
self.mye1.grid(row = 0, column = 1, padx = 10)
self.mye2 = Entry(myroot, width=15, show= '*')
self.mye2.grid(row = 1, column = 1, padx = 10)

def mydisplay():
    print("The username is: " + self.mye1.get())
    print("The password is: " +self.mye2.get())

    self.mybtn = Button(myroot, text = 'Login', command = mydisplay, font = ('Calibri',12))
    self.mybtn.grid(row = 2, columnspan = 3)

if __name__ == "__main__":
    myroot = Tk()
    myobj = MyLogin(myroot)
    myroot.title('Login Page')
    myroot.geometry('200x120')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 4.2*:

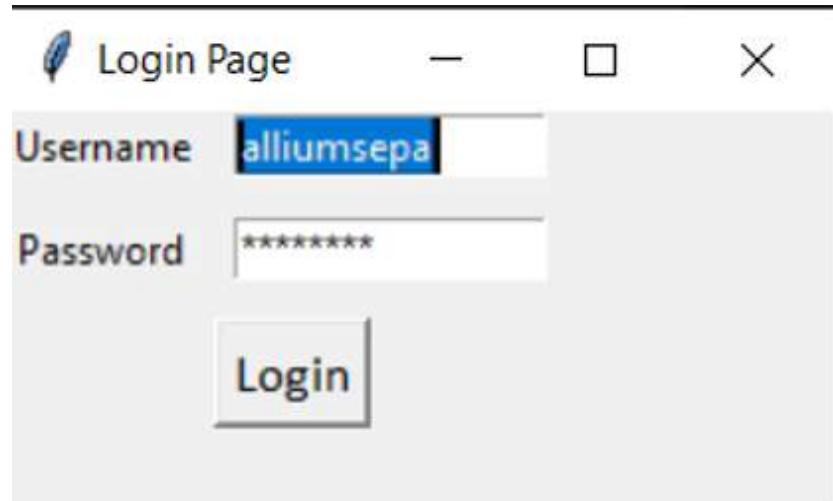


Figure 4.2: Output

Note: The preceding code is covered in Program Name: Chap4_Example2.py

Output when the Login button is clicked:

The username is: alliumsepa

The password is: hello123

In the above code, we have written the password using an asterisk sign by setting it as '*'. This option is used when we are entering very confidential data. Moreover, when the text written inside the **Entry** widget is selected, the **selectborderwidth** is set to 3. If we remove the **selectborderwidth** option, which is by default 1, then observe the output shown in [Figure 4.3](#):

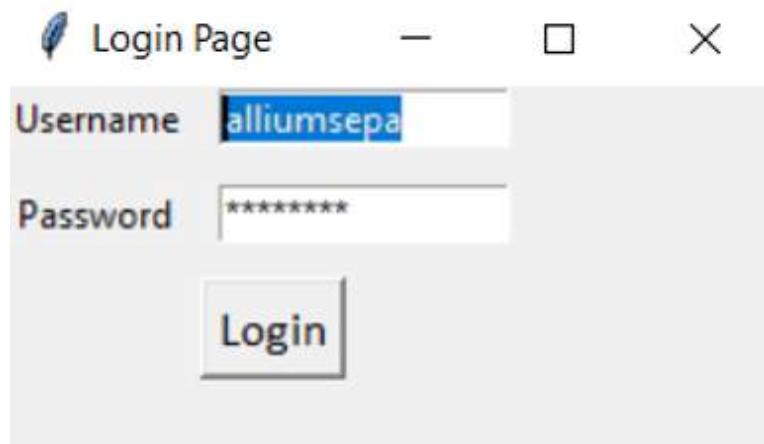


Figure 4.3: Output when selectborderwidth option is removed

Now, we shall see the most commonly used methods with this widget, which are as follows:

- **delete(first, last=None):** This method will delete the specified characters inside the entry widget, starting with the one at the index up to, but not including, the character at the last position.

```
from tkinter import *

class MydeleteExample(Tk):
    def __init__(self):
        super().__init__()
        self.title('MyDelete Example')
        self.myel= Entry(self,font = ('Arial',12),width = 30, bd = 5)
        self.myel.pack(side = LEFT)

        self.button1 = Button(self, text="Delete the text", command=lambda: mydelete(self,self.myel))
        self.button1.pack(pady = 32)

    def mydelete(self, myentry):
        myentry.delete(first=0,last=15)

if __name__ == "__main__":
    myroot = MydeleteExample()
    myroot.geometry('400x100')
    myroot.mainloop()
```

Output:

The output can be seen in [Figure 4.4](#):

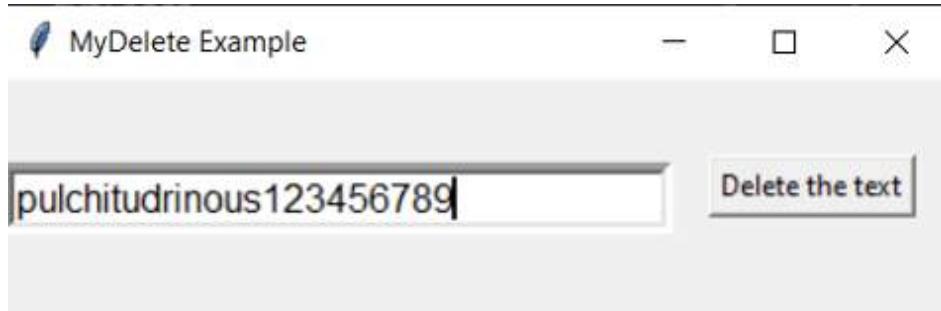


Figure 4.4: Output

The output when **Delete the text** button is clicked, can be seen in [Figure 4.5](#):



Figure 4.5: Output when Delete the text button is clicked

Note: The preceding code is covered in Program Name: [Chap4_Example3.py](#)

From the above code, we can see that the characters starting from index 0 to index 14 will be deleted from the Entry widget on pressing the **Delete the text** button.

- **get()**: This method will return the current text as a string written inside the **Entry** widget.
- **icursor(index)**: This method will change the insertion cursor position. The index of the character is to be specified before which the cursor is to be placed.
- **insert(index,mystr)**: This method will insert the specified string **mystr** before the character is placed at the specified index.

```

from tkinter import *

class MyCursorPosition(Tk):
    def __init__(self):
        super().__init__()
        self.title('MyCursorPosition Example')
        self.myel= Entry(self,font = ('Arial',12),width = 20, bd = 5)
        self.myel.pack(side = LEFT)
        self.myel.focus()
        self.myel.insert(0,'Demonstration')
        self.myel.icursor(0)

        self.button1 = Button(self, text="Position the cursor",
                             command=lambda: myposition(self,self.myel))
        self.button1.pack(pady = 32)

    def myposition(self, myentry):
        myentry.icursor(3)

if __name__ == "__main__":
    myroot = MyCursorPosition()
    myroot.geometry('400x100')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 4.6*:



Figure 4.6: Output

The output when the button **Position the cursor** is clicked, can be seen in [**Figure 4.7**](#):



Figure 4.7: Output when the button Position the cursor is clicked

Note: The preceding code is covered in Program Name: [**Chap4_Example4.py**](#)

In the above code, the specified string ‘Demonstration’ is inserted before the character at the given index 0. By default, the cursor is positioned at index 0 and when the button **Position the cursor** is clicked, the cursor will be placed before the character index position 3.

- **index(index):** This method will place the cursor written at the specified index to the left of the character.
- **select_adjust(index):** This method will include the character selection present at the specified index.

```

from tkinter import *

class MyIndex_Select_adjust(Tk):
    def __init__(self):
        super().__init__()
        self.title('MyIndex and Select_adjust Example')
        self.mye1= Entry(self,font = ('Arial',12),width = 20, bd = 5)
        self.mye1.pack(side = LEFT)
        self.mye1.focus()
        self.mye1.insert(0,'Demonstration')
        self.mye1.icursor(0)

        self.button1 = Button(self, text="Index", command=lambda: myindex(self,self.mye1))
        self.button1.pack(pady = 12)

        self.mybtn2 = Button(self, text="select_adjust", command=lambda: myselect_adjust(self,self.mye1))
        self.mybtn2.pack(pady = 10)

    def myindex(self, myentry):
        myentry.icursor(self.mye1.index(6))

    def myselect_adjust(self, myentry):
        myentry.select_adjust(5)

if __name__ == "__main__":
    myroot = MyIndex_Select_adjust()
    myroot.geometry('400x100')
    myroot.mainloop()

```

Default output when run:

The output can be seen in *Figure 4.8*:



Figure 4.8: Output

The output when **Index** button is clicked, can be seen in [Figure 4.9](#):



Figure 4.9: Output when Index button is clicked

The output when **select_adjust** button is clicked, can be seen in [Figure 4.10](#):

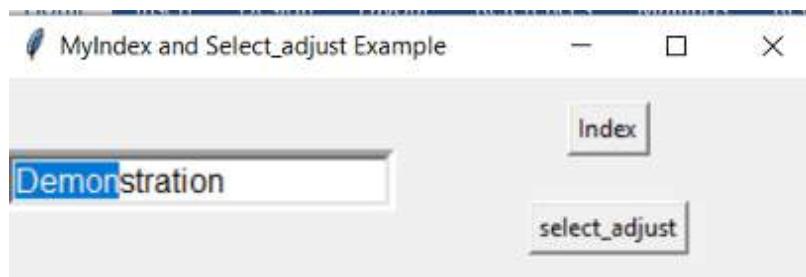


Figure 4.10: Output when select_adjust button is clicked

Note: The preceding code is covered in Program Name: [Chap4_Example5.py](#)

In the above example, we can see that when the **Index** button is clicked, the cursor is placed to the left of the character written at the specified index 6. When the **select_adjust** button is clicked, it will allow the selection of the characters highlighted in blue color at a specified index.

- **select_from(index):** This method will set the index position to anchor the character index selection.

- **select_clear()**: This method will clear the selection if some selection has been done else it has no effect.
- **select_range(start, end)**: This method will select the text in the **Entry** widget between the specified range, that is, the text will be selected at the start index up to, but not including the character at the end index position.
- **select_to(index)**: This method will select all the characters from the anchor position, that is, from the beginning, to the specified index, but not including the character at the given index position.
- **select_present()**: This method will return True if there is some text selected in the Entry widget else returns False.

```

from tkinter import *

class MySelectMethods(Tk):
    def __init__(self):
        super().__init__()
        self.title('MyIndex and Select_adjust Example')
        self.
mye1= Entry(self,font = ('Arial',12),width = 20, bd = 5)
        self.mye1.pack(side = LEFT)
        self.mye1.focus()
        self.mye1.insert(0,'Demonstration')
        self.mye1.icursor(0)
        self.mye1.select_clear()

        self.button1 = Button(self, text="select_
to", command=lambda: myselect_to(self,self.mye1))
        self.button1.pack(pady = 5)

        self.mybtn2 = Button(self, text="select_

```

```
from", command=lambda: myselect_from(self,self.mye1))
    self.mybtn2.pack(pady = 5)

    self.mybtn3 = Button(self, text="select_
range", command=lambda: myselect_range(self,self.mye1))
    self.mybtn3.pack(pady = 5)

    self.mybtn4 = Button(self, text="select_
clear", command=lambda: myselect_clear(self,self.mye1))
    self.mybtn4.pack(pady = 5)

    self.mybtn5 = Button(self, text="select_
present", command=lambda: myselect_present(self,self.mye1))
    self.mybtn5.pack(pady = 5)

def myselect_to(self, myentry):
    myentry.select_to(4)

def myselect_from(self, myentry):
    myentry.select_from(1)

def myselect_range(self, myentry):
    myentry.select_range(6,9)

def myselect_clear(self, myentry):
    myentry.select_clear()

def myselect_present(self, myentry):
    print(myentry.select_present())

if __name__ == "__main__":
    myroot = MySelectMethods()
    myroot.geometry('400x200')
    myroot.mainloop()
```

Default output when run:

The output can be seen in *Figure 4.11*:

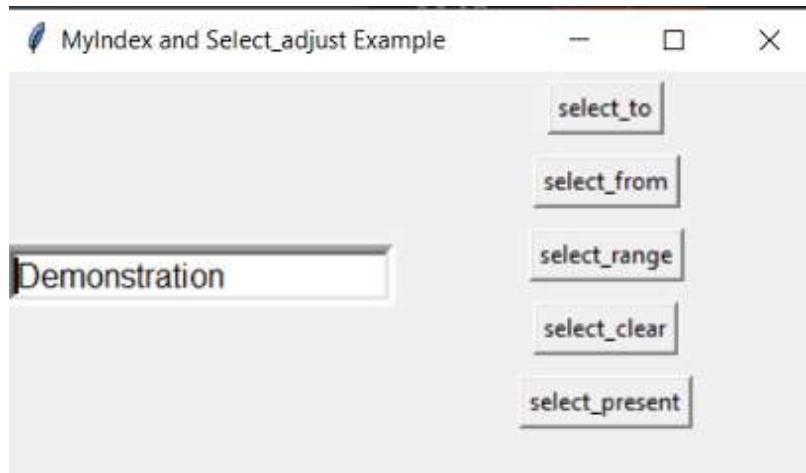


Figure 4.11: Output

The output when the **select_to** button is clicked, can be seen in *Figure 4.12*:

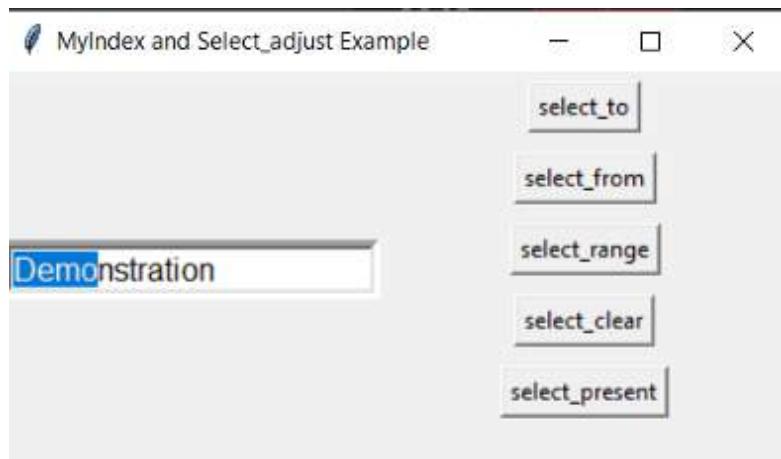


Figure 4.12: Output when select_to button is clicked

We can see from the above code that when the **select_to** button is clicked, all the characters from the anchor position, that is, from the beginning 0 to the specified index 4 but not including the character at the given index position, that is, till index 3, will be selected.

Output when the select_range button is clicked:

Refer to *Figure 4.13*:

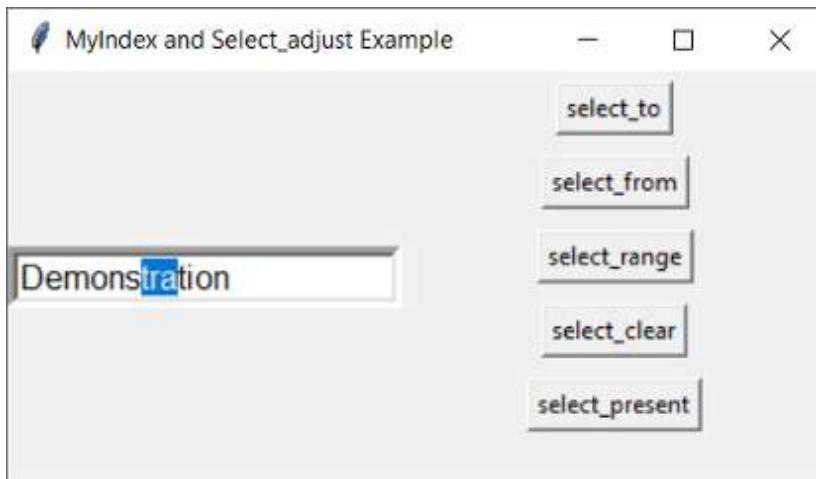


Figure 4.13: Output when **select_range** button is clicked

We can see that when the **select_range** button is clicked, characters from index position 6 to index position 8 are selected. That is why we can view that characters ‘tra’ are highlighted in blue color.

The output when **select_from** button is first clicked followed by clicking of **select_to** button, can be seen in the following *Figure 4.14*:

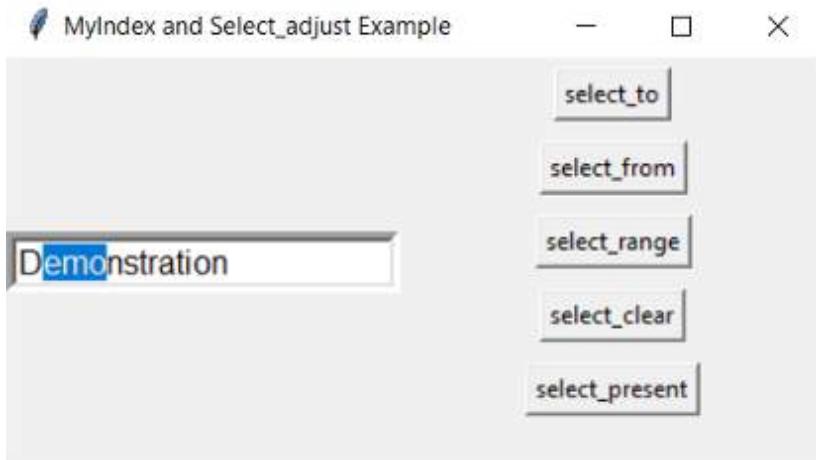


Figure 4.14: Output when **select_from** button is clicked

When the **select_from** button is first clicked, the anchor index position is set to the character selected by index 1. Now, when we again click the **select_to**

button, the index position, and the text from the anchor position, that is, 1 up to index position 3 will be selected as shown in the following figures.

Output when the **select_present** button is clicked:

Refer to *Figure 4.15*:

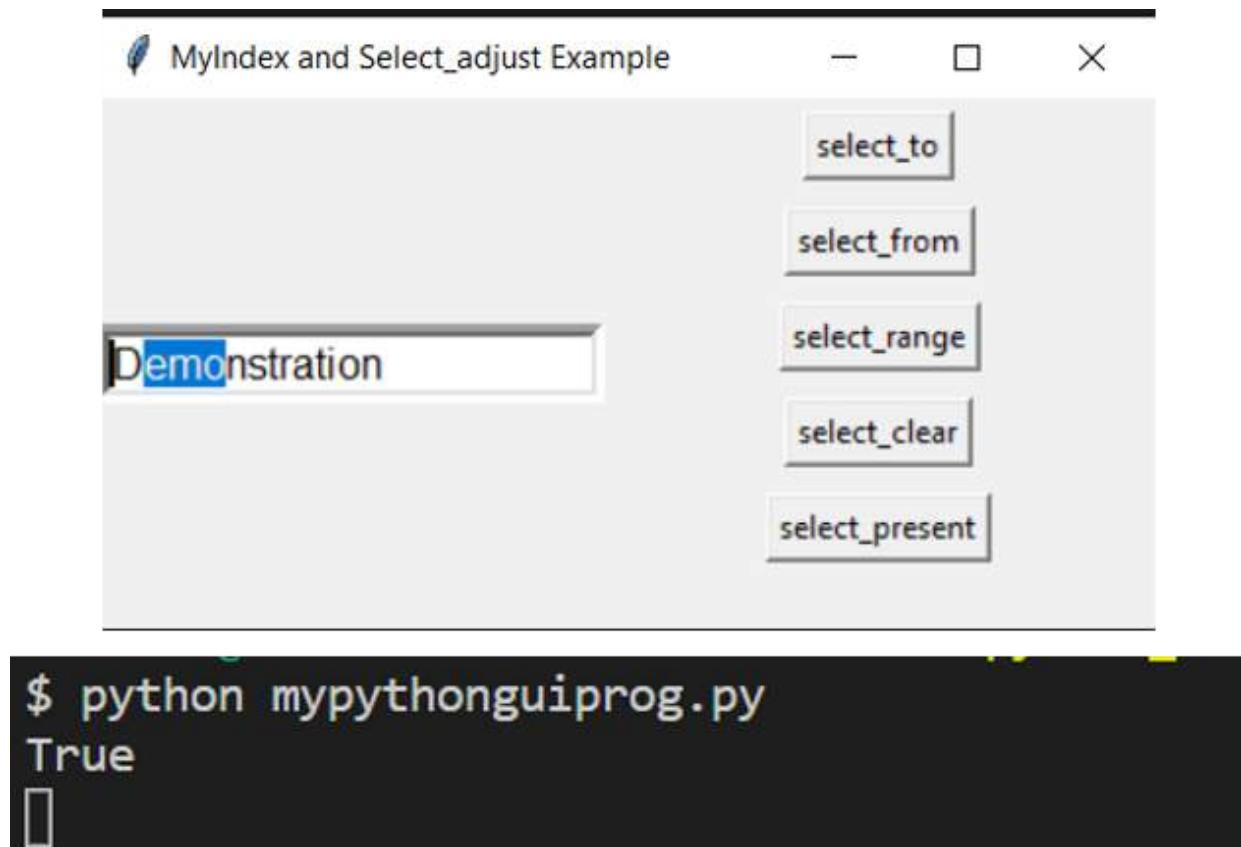


Figure 4.15: Output when select_present button is clicked

When the **select_present** button is clicked, True is returned since there is a selection in the **Entry** widget as shown.

The output when the **select_clear** button is clicked can be shown in *Figure 4.16*:

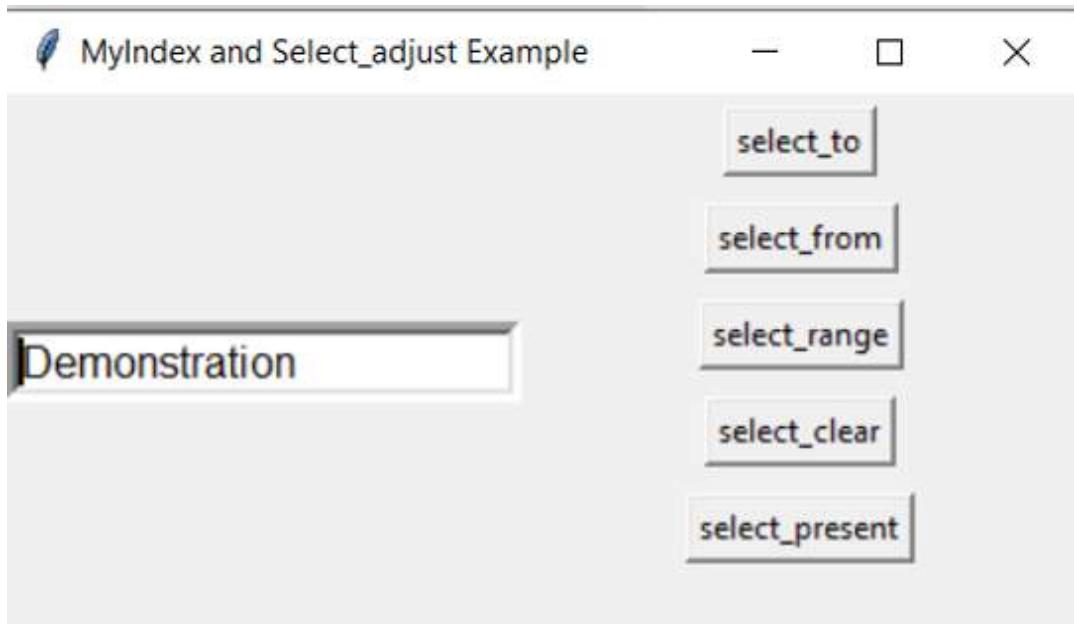


Figure 4.16: Output when `select_clear` button is clicked

Note: The preceding code is covered in Program Name: `Chap4_Example6.py`

When the `select_clear` button is clicked, the selection is cleared without deleting the content as shown.

Now, if we will click the `select_present` button, we will return `False`.

- `xview_scroll(number, what)`: This method will scroll the `Entry` widget horizontally. The first argument number must be either in UNITS or PAGES where scrolling can be done by character widths or by chunks to the size of the `Entry` widget. The scrolling is done from left to right when positive or from right to left when negative.
- `xview(index)`: This method will link a horizontal scrollbar in the `Entry` widget, as shown:

```
from tkinter import *

class MyScrollbarEntry(Tk):
    def __init__(self):
        super().__init__()
        mysobj_scroll = Scrollbar(self,orient = 'horizontal')
        mye1 = Entry(self,xscrollcommand = mysobj_scroll.
set, font = ('Calibri',12))
        mye1.focus()
        mye1.pack(side= 'bottom', fill = X)
        mysobj_scroll.pack(fill = X)
        mysobj_scroll.config(command = mye1.xview)

        mye1.insert(0, 'We should follow social distancing when we are going outside from our home. It is mandatory to follow.')

if __name__ == "__main__":
    myroot = MyScrollbarEntry()
    myroot.geometry('400x200')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 4.17*:

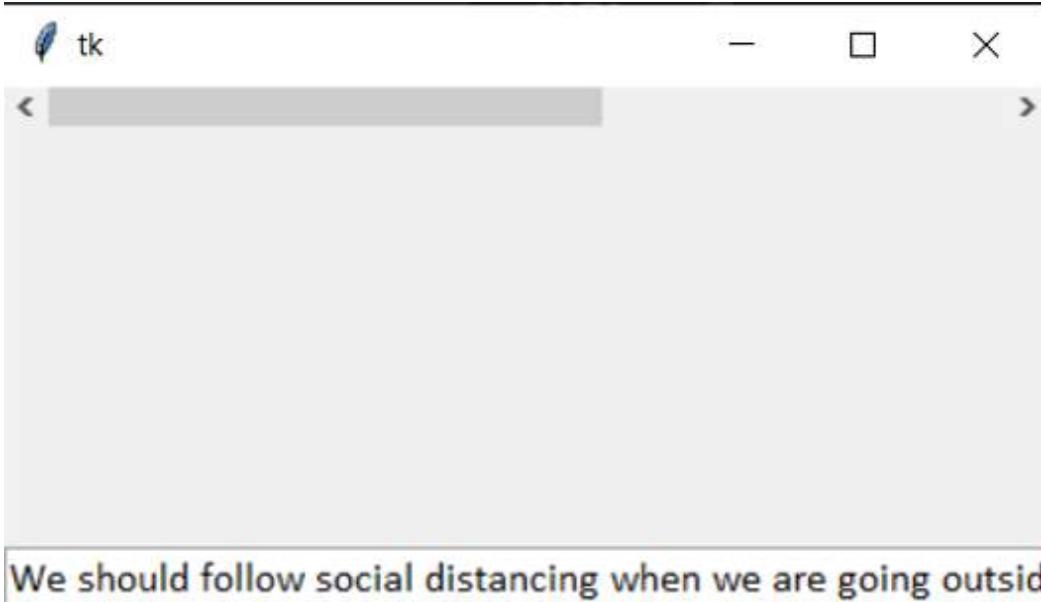


Figure 4.17: Output

Note: The preceding code is covered in Program Name: Chap4_Example7.py

In this code, we are linking a horizontal scrollbar to an **Entry** widget. We can see a scrollbar at the top of the **Entry** widget.

Validation in the **Entry** widget

There will be some cases where the text written inside an **Entry** widget is to be checked to make sure that it is valid according to some rule. Such validation on an **Entry** widget can be done as follows:

- A callback function is defined, which will check the text in the **Entry** and will return True if valid, and otherwise, it will return False. The text will be unchanged if the callback returns False.
- Next is to register the callback function. A character string is returned which will be used to call the function.
- Then, validate the input in the **Entry** widget by calling the callback function. The options used are described as follows.

validate: This option will be used to specify when to call the callback function to validate the input. The values of the validate command are:

- **none**: If the validation is set to None, then no validation occurs. It is the default mode.
- **focus**: If the validate is set to focus, the **validatecommand** is called twice when the Entry widget receives focus and when the focus is lost.
- **focusin**: The **validatecommand** is called when the widget has focus.
- **focusout**: The **validatecommand** is called when the widget has lost focus.
- **key**: The **validatecommand** is called whenever any input from the keyboard changes the widget's contents.
- **all**: The **validatecommand** will be called in all the above cases.

validatecommand: This option is used to specify the callback function, that is, what arguments our callback function would like to receive. The callback function is needed to know what text appears in the **Entry** widget, but this callback function will not be called directly but rather by a variable that is passed and registered in the previous steps. A number of items of information are also provided to the callback via substitution codes which are as follows:

- **%d**: This substitution code is an action type that occurred on the widget, 0 for attempted deletion, 1 for attempted insertion, and -1 for focus, forced, or textvariable validation.
- **%i**: Whenever any text is inserted or deleted, this substitution code will be an index of the beginning of deletion or insertion. If the callback is due to focusin, focusout, or change in the textvariable, it will be -1.
- **%P**: This substitution code will be the value the widget will have if the change is allowed.
- **%s**: This substitution code denotes the current text in the Entry widget prior to editing.
- **%S**: This substitution code denotes the text being inserted or deleted.
- **%v**: This substitution code denotes the validation type currently set.

- **%V**: This substitution code denotes the validation type for which the callback is triggered like focusin, focusout, key, forced, or textvariable.
- **%w**: This substitution code denotes the widget's name.

Let us see an example.

It should be kept in mind that in the above example, we are using classes. So, we will be using methods instead of functions. The same code can be replicated without using classes. So, we will be using the term functions. Refer to the following code:

```
from tkinter import *
class MyValidate(Tk):
    def __init__(self):
        super().__init__()
        self.myL0 = Label(self, text= 'Enter the number:', fg='Magenta', font = ('Arial',12))
        self.myL0.place(x = 10, y = 30)

        self.myE1 = Entry(self, font = ('Helvetica',12))
        self.myE1.place(x = 150, y = 30)

        self.myL1 = Label(self, text= '', fg='Red')
        self.myL1.place(x = 70, y = 50)

        self.myreg = self.register(self.mycallback) # V1
        self.invalidcmd = self.register(self.myinvalid_name) # V2
        self.myE1.config(validate ="key", validatecommand =(self.
myreg, '%P'), invalidcommand = (self.invalidcmd, '%S')) # V3
```

```
def mycallback(self, myinp):
    if myinp.isdigit():# C1
        print(mynp)
        self.myl1.config(text=' ')
        return True

    elif myinp is "": # C2
        print(mynp)
        self.myl1.config(text=' ')
        return True

    else: # C3
        print(mynp)
        return False

def myinvalid_name(self, myCh):
    self.myl1.config(text=(f'Invalid character {myCh} \n name can only have numbers'), font = ('Verdana',10))

if __name__ == "__main__":
    myroot = MyValidate()
    myroot.geometry('300x100')
    myroot.mainloop()
```

Output:

Refer to *Figure 4.18*:

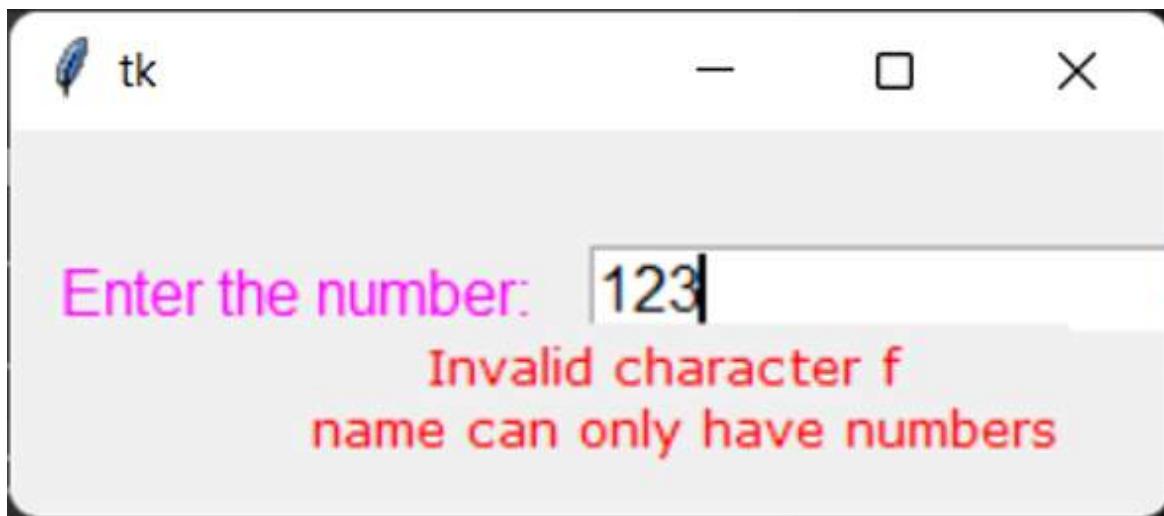


Figure 4.18: Output

Note: The preceding code is covered in Program Name: Chap4_Example8.py

Output at the console:

```
1
12
123
123f
123g
12
1
```

In the preceding code, we have created 2 labels and 1 Entry widget and positioned them in the parent widget. In V1, a string is returned which will be assigned to a variable **myreg** and the position will call the **mycallback** method.

In V2, a string is returned which will be assigned to a variable **invalidcmd** and will call the **myinvalid_name** method.

In V3, we are using the options **validate**, **validatecommand**, and **invalidcommand**. The key option in validate will specify that validation occurs whenever any input from the keyboard changes the **Entry** widget

options. The validate command will specify the **mycallback** method and is called by passing a variable **myreg**. The **invalidcommand** is optional and will specify the **myinvalid_name** method and is called by passing a variable **invalidcmd**. The **myinvalid_name** method will be called when the **validatecommand** returns False when we are entering alphabets.

In C1, the **mycallback** method returns True when we enter any digits from our keyboard as its value is allowed in the entry widget.

In C2, the digits can be erased by using the backspace key.

In C3, the **mycallback** method returns False, when the user enters the alphabet from the keyboard and its value is not allowed in the **Entry** widget.

The input is getting displayed on the console whether any insertion and deletion of digits or alphabets are entered from the keyboard. The digits are added and can be erased.

The same code can be written without using any class. So, here we can use the term function instead of method.

```
from tkinter import *  
  
myroot = Tk()  
myroot.geometry('300x100')
```

```

myl0 = Label(myroot, text= 'Enter the number:', fg=
'Magenta', font = ('Arial',12))
myl0.place(x = 10, y = 30)
mye1 = Entry(myroot, font = ('Helvetica',12))
mye1.place(x = 150, y = 30)
myl1 = Label(myroot, text= ' ', fg='Red')
myl1.place(x = 70, y = 50)

def mycallback(myinp):
    if myinp.isdigit():
        print(myinp)
        myl1.config(text=' ')
        return True

    elif myinp is "":
        print(myinp)
        myl1.config(text=' ')
        return True

    else:
        print(myinp)
        return False

def myinvalid_name(myCh):
    myl1.config(text=(f'Invalid character {myCh} \n name can only have numbers'), font = ('Verdana',10))

myreg = myroot.register(mycallback)
invalidcmd = myroot.register(myinvalid_name)
mye1.config(validate ="key", validatecommand =(myreg, '%P'), invalidcommand = (invalidcmd, '%S'))

myroot.mainloop()

```

Note: The preceding code is covered in Program Name: Chap4_Example8_2.py

We will get the same output.

tkinter Scrollbar widget

This widget will add the scrolling capability to the various widgets such as ListBox, Canvas, Entry, and Text. The vertical scrollbar can be implemented in ListBox, Canvas, and Text widgets whereas the horizontal scrollbar can be implemented in the Entry widget. The content is rolled through vertically or horizontally.

The syntax is:

```
mysc1= Scrollbar(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, activebackground, cursor, command, elementborderwidth, highlightcolor, highlightbackground, highlightthickness, orient, jump, repeatdelay, repeatinterval, width, takefocus, and troghcolor.

We have seen most of the options but some undiscussed options are as follows:

- **command**: This option will associate with a function whenever the scrollbar is moved by the user.
- **elementborderwidth**: This option will specify the border width around the arrow heads/cursor points and slider. The default value of **elementborderwidth** is -1 and can be set as per need.
- **orient**: This option will allow the orientation scrollbar and can be set to either HORIZONTAL or VERTICAL.
- **jump**: This option will control the scroll jump behavior. The default value is 0 where every small slider drag will cause the command

callback to be called. The callback is not called when set to 1, until the user releases the mouse button.

- **repeatdelay**: This option whose default duration is 300msec will allow controlling the duration of the button1 to be held down in the trough before the slider will start moving in that direction repeatedly.
- **repeatinterval**: This option is for repeating the slider interval whose default value is 100msec.
- **takefocus**: This option will allow tabbing the focus through the scrollbar widget. We can set this option to 0 when not needed.
- **troughcolor**: This option will allow setting the trough color.

The methods used in this widget are as follows:

- **get**: This method will represent the current scrollbar position and will return the numbers a and b, where a represents the slider top or left edge for horizontal or vertical scrollbars and b represents the slider bottom or right edge.
- **set(first, last)**: This method will allow connecting the scrollbar to the other widget. The widget **xscrollcommand** or **yscrollcommand** will be set to the scrollbar's set method.
- **pack**: This method will set the slider alignment.

Now, we shall see some examples of scrollbar with other widgets.

Scrollbar attached to Listbox

Refer to the following code:

```

from tkinter import *

class Scrollbar_ListBox(Tk):
    def __init__(self):
        super().__init__()

        self.mysclbar = Scrollbar(self)# scrollbar creation and attaching to the main window
        self.mysclbar.pack(side=RIGHT, fill="y") # scrollbar added to the window right side

        self.mylistbox = Listbox(self)# listbox creation and attaching to the main window
        self.mylistbox.config(yscrollcommand=self.mysclbar.set) # scrollbar attached to the listbox . for vertical scroll used yscrollcommand

        for loop in range(50): # insert elements from 0 to 49 in the listbox
            self.mylistbox.insert(END, str(loop))

        self.mylistbox.pack(side="left", fill=BOTH) # listbox added to the window left side
        self.mysclbar.config(command=self.mylistbox.yview) # for need of vertical view settings scrollbar command option to listbox.yview method

    if __name__ == '__main__':
        myroot = Scrollbar_ListBox() # creating an instance of Scrollbar_Listbox
        myroot.mainloop() # infinite loop to run the application

```

Output:

The output can be seen in the following *Figure 4.19*:

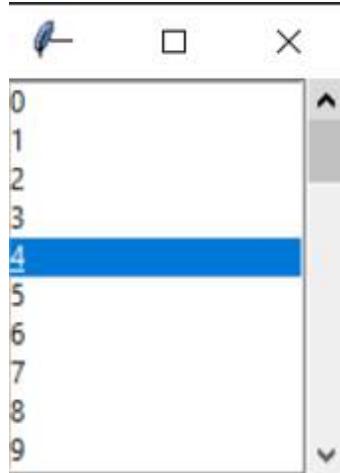


Figure 4.19: Output

Note: The preceding code is covered in Program Name: Chap4_Example9.py

Scrollbar attached to Text

Refer to the following code:

```
from tkinter import *

class Scrollbar_Text(Tk):
    def __init__(self):
        super().__init__()

        self.mysclbar = Scrollbar(self)# scrollbar creation and at-
        taching to the main window
        self.mysclbar.pack(side=RIGHT, fill=Y) # scrollbar add-
```

```
ed to the window right side

    self.sclhbar = Scrollbar(self,orient = HORIZONTAL)
    self.sclhbar.pack(side = BOTTOM,fill = X)

    self.mytxt = Text(self,
                      width = 600,
                      height = 600,
                      yscrollcommand=self.mysclbar.set,
                      xscrollcommand=self.sclhbar.set,
                      wrap = NONE) # creation of text-
box and both horizontal and vertical scrollbars are at-
tached to the textbox

    self.mytxt.pack(expand = 0, fill=BOTH)

    # horizontal elements
    for loop in range(26): # insertele-
ments from 0 to 49 in the text
        self.mytxt.insert(END, str(loop) + '\t')
    # vertical elements
    for loop in range(50): # insertele-
ments from 0 to 49 in the text
        self.mytxt.insert(END, str(loop) + '\n')

    self.sclhbar.config(command=self.mytxt.
xview)# for need of horizontal view settings scrollbar command op-
tion to textbox.xview method
    self.mysclbar.config(command=self.mytxt.
yview) # for need of vertical view settings scrollbar command op-
tion to textbox.yview method

if __name__ == '__main__':
    myroot = Scrollbar_Text() # creating an instance of Scrollbar_
Text
    myroot.geometry('300x300')
    myroot.mainloop() # infinite loop to run the application
```

Output:

The output can be seen in the following *Figure 4.20*:

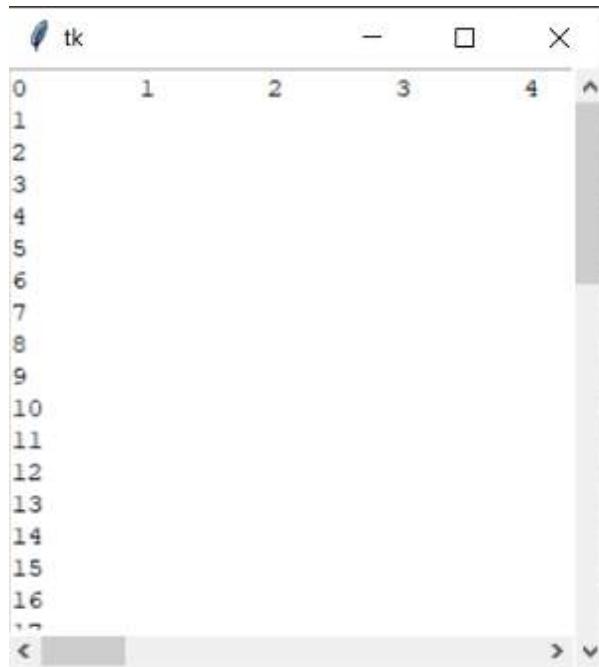


Figure 4.20: Output

Note: The preceding code is covered in Program Name: Chap4_Example10.py

Scrollbar attached to Canvas

The output can be seen in the following *Figure 4.21*:

```

from tkinter import *

class Scrollbar_Canvas(Tk):
    def __init__(self):
        super().__init__()

        mycanvas = Canvas(self, width=150, height=50)
        mycanvas.create_oval(20, 20, 80, 80, fill="red")
        mycanvas.create_oval(200, 200, 280, 280, fill="blue")
        mycanvas.grid(row=0, column=0)

        myscroll_x = Scrollbar(self, orient="horizontal", command=-
mycanvas.xview)
        myscroll_x.grid(row=1, column=0, sticky=EW)

        myscroll_y = Scrollbar(self, command=mycanvas.yview)
        myscroll_y.grid(row=0, column=1, sticky=NS)

        mycanvas.configure(scrollregion=mycanvas.bbox-
("all")) # will return the rectangular coordinates fit-
ting the whole canvas content. Here the position of 2 corners of a
rectangle is described which is a scroll region. It is a 4 valued
tuple.

if __name__ == '__main__':
    myroot = Scrollbar_Canvas() # creating an instance of Scroll-
bar_Entry
    myroot.geometry('200x150')
    myroot.mainloop() # infinite loop to run the application

```

Output:

The output can be seen in the following *Figure 4.21*:

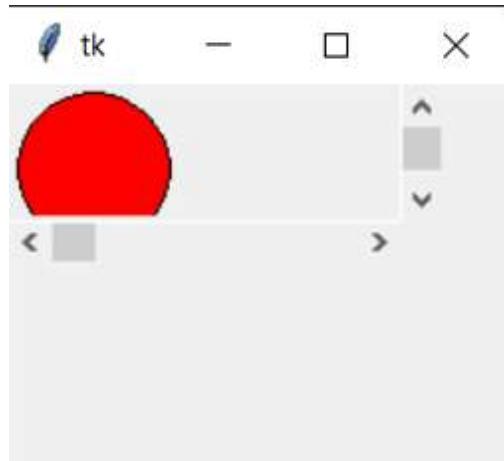


Figure 4.21: Output

Note: The preceding code is covered in Program Name: Chap4_Example11.py

Scrollbar attached to Entry

Refer to the following code:

```
from tkinter import *

class Scrollbar_Entry(Tk):
    def __init__(self):
        super().__init__()
```

```

        self.sclhbar = Scrollbar(self,orient = HORIZONTAL)
        self.sclhbar.pack(side = BOTTOM,fill = X)

        self.myel = Entry(self,xscrollcommand=self.sclhbar.
set) # creation of entry and horizontal scrollbars is attached
        self.myel.pack(expand = 0, fill=BOTH)

        # horizontal elements
        for loop in range(26): # insert elements from 0 to 25 in the listbox
            self.myel.insert(END, str(loop) + '\t')
        self.sclhbar.config(command=self.myel.
xview)# for need of horizontal view settings scrollbar command option to entry.xview method

if __name__ == '__main__':
    myroot = Scrollbar_Entry() # creating an instance of Scrollbar_
Entry
    myroot.geometry('300x100')
    myroot.mainloop() # infinite loop to run the application

```

Output:

The output can be seen in the following *Figure 4.22*:



Figure 4.22: Output

Note: The preceding code is covered in Program Name: Chap4_Example12.py

tkinter Spinbox widget

This widget will allow the user to choose from some fixed range of values. It is an alternative to the **Entry** widget which provides values range to the user, from where the user can select the one. It specifies the values to be allowed which can be either a range or a tuple.

The syntax is as follows:

```
mysp1= Spinbox(myroot, options...)
```

where

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, activebackground, cursor, command, disabledbackground, disabledforeground, font, fg, format, from_, relief, justify, repeatdelay, state, repeatinterval, textvariable, to, values, validate, validatecommand, wrap, width, and xscrollcommand.

We have seen most of the options but some undiscussed options are as follows:

- **command**: This option will call the function or method whenever there is the movement of the scrollbar. So, by using the command option we are adding functionality to the above widget.
- **format**: This option will be used for formatting the string and there is no default value.
- **from_**: This option displays the minimum limit value which will show the widget starting range.
- **repeatdelay**: This option will control the button autorepeat and is given in milliseconds.

- **repeatinterval**: This option is similar to **repeatinterval** and is given in milliseconds.
- **textvariable**: This option will have no default value and is a control variable that will control the widget behavior text.
- **to**: This option displays the maximum limit value of the widget.
- **validate**: This option will represent the validation mode and its default value is None.
- **validatecommand**: This option will represent the validation callback and there is no default value.
- **wrap**: This option will be wrapping the up and down button of the above widget.
- **xscrollcommand**: This option will be set to the above widget **set()** method for scrolling the widget horizontally.

Some of the commonly used methods in the above widget are as follows:

- **delete(startindex [,endindex])**: This method will delete the characters within the specified range.
- **get(startindex [,endindex])**: This method will get the characters within the specified range.
- **identify(x,y)**: This method will identify the widget's element within the specified range.
- **index(index)**: This method will get the absolute value of the given index.
- **insert(index [string]...)**: This method will insert the string at the specified index.
- **selection_clear()**: This method will clear the selection.
- **selection_get()**: This method will return the selected text and will raise an exception if there is no selection.

We shall see some examples for better understanding.

Let us create a spinbox:

```
from tkinter import *

myroot = Tk()
myroot.geometry('250x100')
myroot.title('SpinBox')

# creation of spinbox
mys1 = Spinbox(font = ('Calibri',15), from_ = 10, to = 20)
mys1.pack()

myroot.mainloop()
```

Output:

The output can be seen in the following *Figure 4.23*:

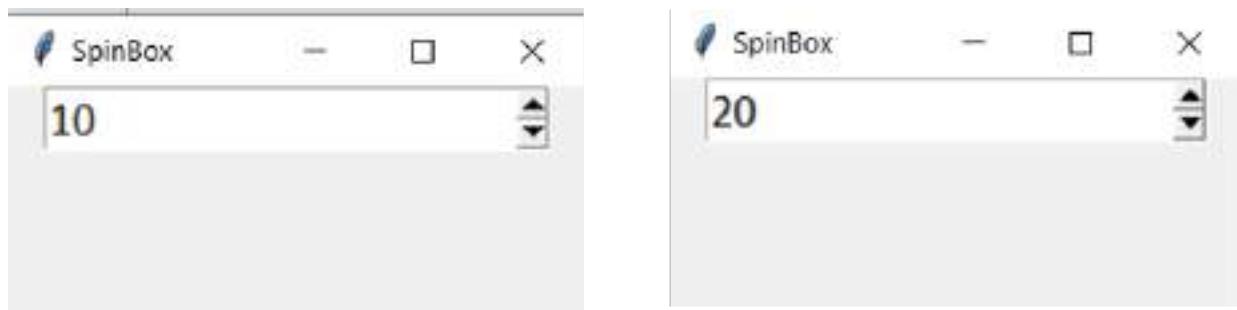


Figure 4.23: Output

Note: The preceding code is covered in Program Name: Chap4_Example13.py

In the above code, we have created a spinbox with a minimum value of 10 and a maximum value up to 20.

Instead of using a range, we can also specify a set of values, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry('250x100')
myroot.title('SpinBox')

# creation of spinbox
mys1 = Spinbox(font = ('Calibri',15), values = (10,35,49,40), bd = 10,
relief = RAISED)
mys1.pack(pady = 10)

myroot.mainloop()
```

Output:

The output can be seen in the following *Figure 4.24*:

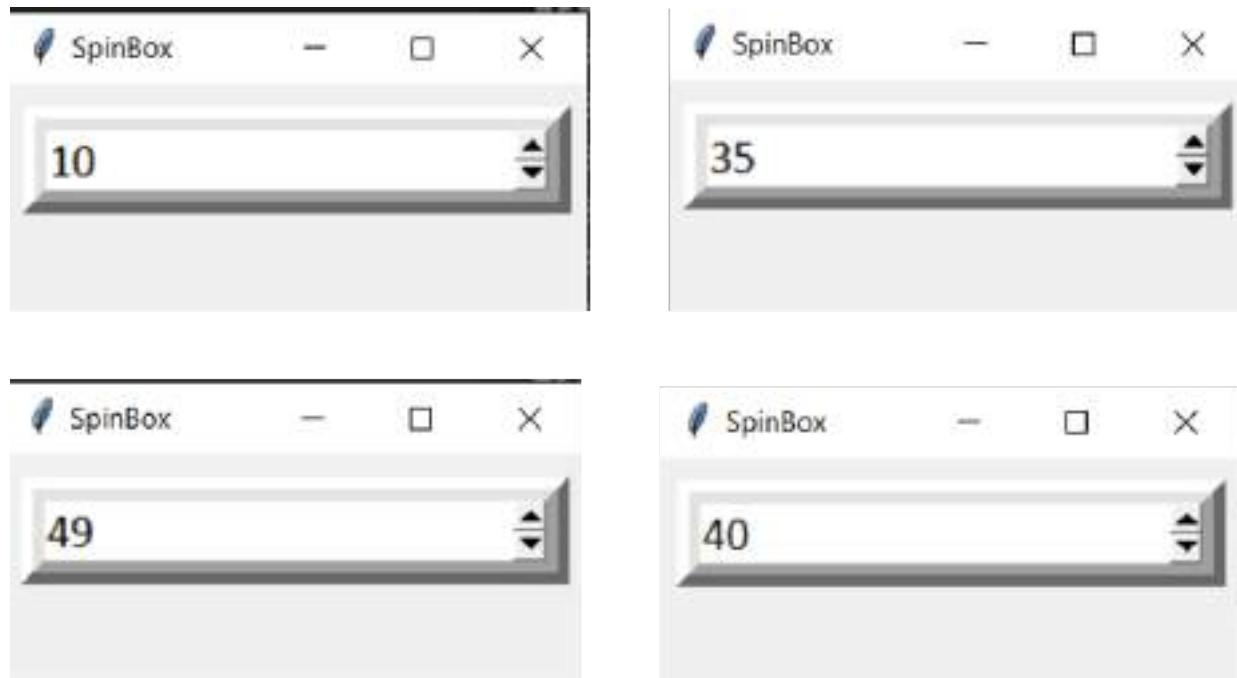


Figure 4.24: Output

Note: The preceding code is covered in Program Name: Chap4_Example14.py

We can also display string values to a spinbox and perform a callback function on the movement of the spinbox, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry('300x300')

# stringvar variable
a1 = StringVar()

# mydisplay function
def mydisplay():
    myroot.configure(bg = a1.get())

# creation of spinbox
mys1 = Spinbox(font = ('Calibri',15), command = mydisplay, values = ['Red','Green','Blue','Violet','Indigo','Magenta','Yellow'], textvariable = a1)
mys1.pack()

myroot.mainloop()
```

Output:

Refer to *Figure 4.25*:

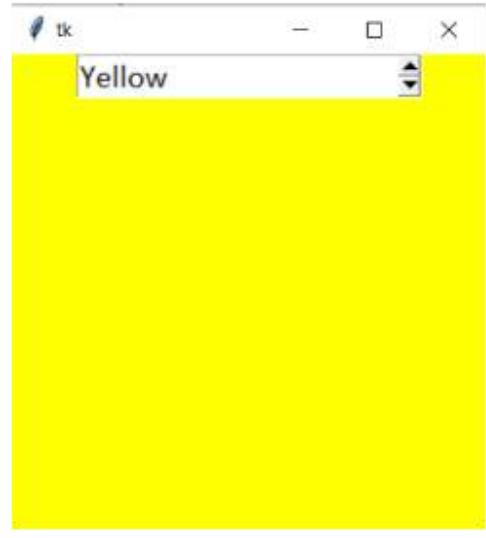
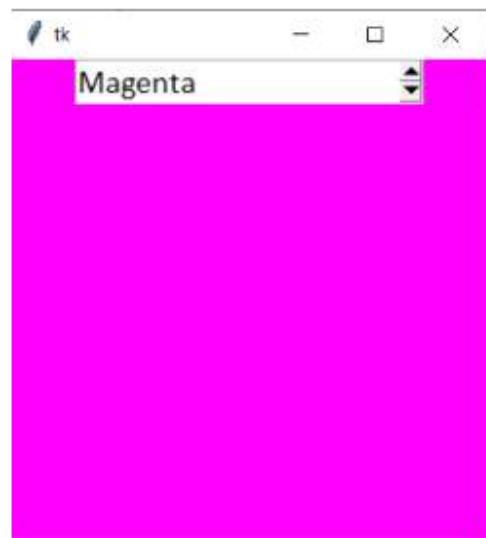
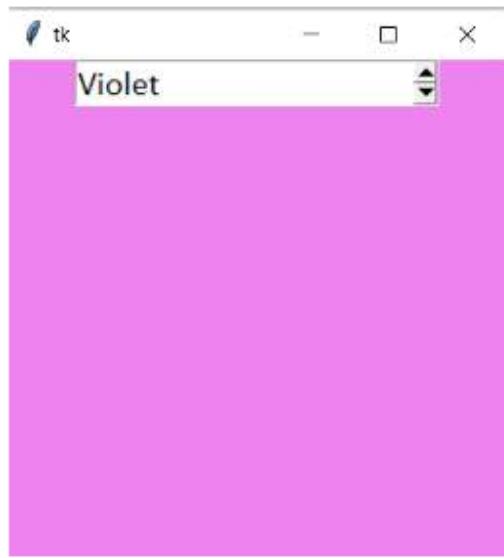
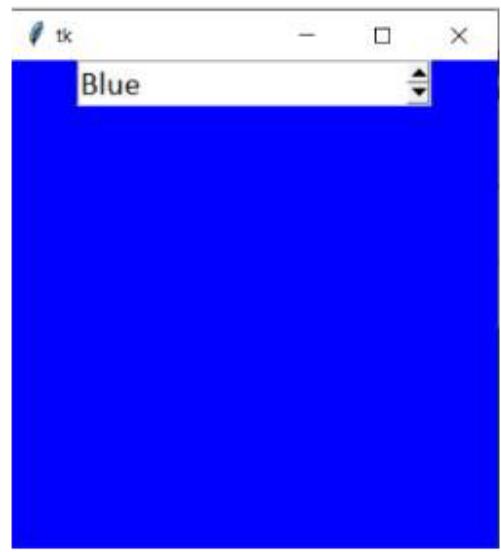
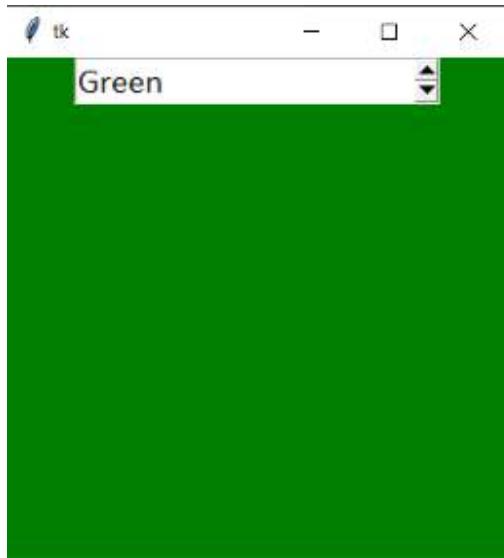


Figure 4.25: Output

Note: The preceding code is covered in Program Name: Chap4_Example15.py

In the above code, we have displayed string values to the spinbox, and on scrollbar movement, we are changing the background color of the parent window.

We can also disable clicking in, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry('250x100')
myroot.title('SpinBox')

# creation of spinbox
mys1 = Spinbox(font = ('Calibri',15), values = (10,35,49,40,50,60),state = 'readonly')
mys1.pack(pady = 10)

myroot.mainloop()
```

Output:

Refer to *Figure 4.26*:

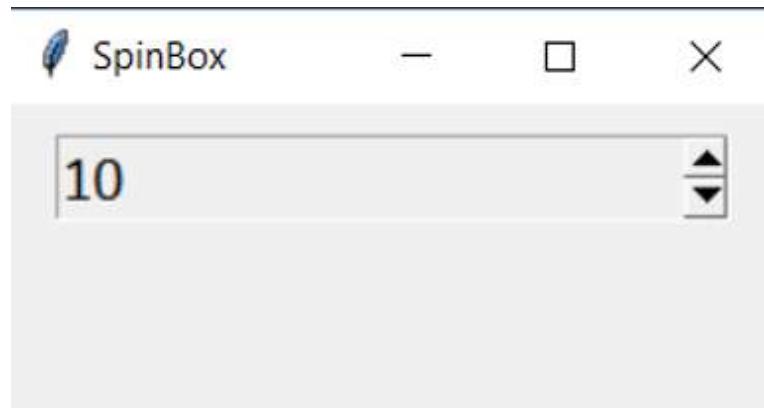


Figure 4.26: Output

Note: The preceding code is covered in Program Name: Chap4_Example16.py

tkinter Scale widget

This widget will allow you to select from a range of numbers by providing a graphical slider object and moving through a slider. Here, we can set the minimum and maximum values.

The syntax is as follows:

```
myscl1= Scale(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, activebackground, command, digits, cursor, font, fg, highlightbackground, highlightcolor, from_, length, label, orient, repeatdelay, resolution, relief, showvalue, sliderlength, state, tickinterval, takefocus, to, variable, troughcolor, and width.

We have seen most of the options but some undiscussed options are as follows:

- **digits**: This option will read the current value via the control variable and is used to specify the digits to be displayed on the scale range.
- **from_**: This option will specify the starting point of the scale range.
- **to**: This option will specify the ending point of the scale range.
- **label**: This option will display a text label of the scale which is shown on the top left and right corners vertically and extreme left and right corners horizontally.
- **orient**: This option's default orientation is horizontal and can be set to horizontal or vertical depending on the scale type.

- **repeatdelay**: This option will define the duration up to which button1 will be held in the trough before the slider starts moving in that direction repeatedly and its default is 300msec.
- **resolution**: This option will allow specifying the slightest change possibly made to the scale value. If set to resolution = 1 and from_ = -2 and to = 2 and the scale will have 5 possible values: -2, -1, 0, +1, and +2.
- **showvalue**: This option will show the current value as text by the slider. It can be suppressed by setting it to 0.
- **sliderlength**: This option will specify the scale length and the default is 30 pixels.
- **state**: This option will represent the DISABLED or ACTIVE scale state.
- **tickinterval**: This option will display the scale values at the set intervals.
- **takefocus**: This option will allow a focusing cycle through scale widgets.
- **variable**: This option displays the control variable to monitor the scale state.
- **troughcolor**: This option displays the color of the trough.

Some of the methods in the above widget are as follows:

- **get()**: This method will return the current scale value.
- **set(value)**: This method will set the scale value.
- **cords(value = None)**: This method will return a screen coordinate corresponding to the given scale value.

We shall see some examples to see how the above widget works:

```

from tkinter import *

myroot = Tk()

# creating a float variable value holder
myv1 = DoubleVar()

#creation of horizontal slider
mys1 = Scale(myroot, from_=0, to=100, orient = HORIZONTAL, length = 200, width = 10,
             sliderlength = 50, label = 'myscale',
             variable = myv1) # default length = 100, width = 15, sliderlength = 30
# setting the scale value to 45
mys1.set(45)
mys1.pack()

def mydisplay():
    # will get the value
    print(myv1.get())
    # will return the coordinates corresponding to the given scale value
    print(mys1.coords(value = myv1.get()))

# creating a button widget
mybtn1 = Button(myroot, text = "GetValue", command = mydisplay, bg = 'LightBlue')
mybtn1.pack(pady = 10)

myroot.title('MyScalewidget')
myroot.geometry("300x200+120+120")
myroot.mainloop()

```

Output:

Refer to *Figure 4.27*:

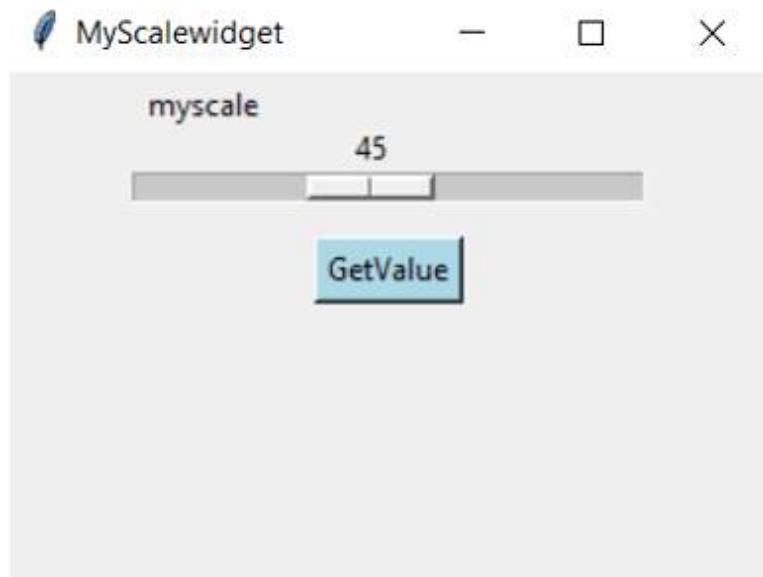
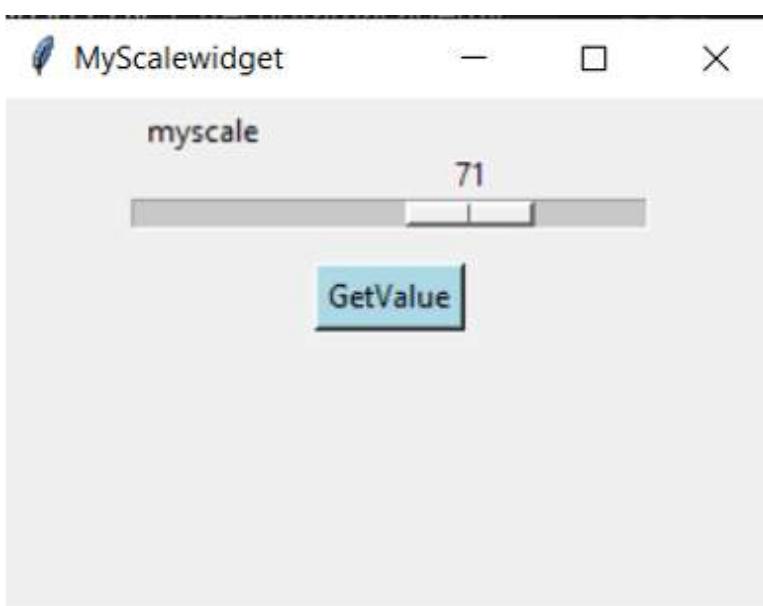


Figure 4.27: Output

The output when **GetValue** button is clicked, can be seen in the following
[**Figure 4.28:**](#)



```
$ python mypythonguiproj.py
71.0
(134, 45)
```

Figure 4.28: Output

Note: The preceding code is covered in Program Name: Chap4_Example17.py

In the above code, we have created a horizontally oriented scale widget with a default value of 45. We can move the slider and when the GetValue button is clicked, the scale value output will be displayed along with the coordinates corresponding to the given scale value.

We can also display the scale orientation vertically, as shown:

```
from tkinter import *

myroot = Tk()

# creating a float variable value holder
myv1 = DoubleVar()

#creation of horizontal slider
mys1 = Scale(myroot, from_=0, to=100, orient = VERTICAL, length = 200, width = 10,
             sliderlength = 50, label = 'MyScale Widget',
             variable = myv1) # default length = 100, width = 15, sliderlength = 30
# setting the scale value to 35
mys1.set(35)
mys1.pack()

def mydisplay():
    # will display the value
    myl1.config(text = 'The scale value is: ' + str(myv1.get()), font = ('Verdana',12))

# creating a button widget
mybtn1 = Button(myroot, text = "GetValue", command = mydisplay, bg = 'LightBlue')
mybtn1.pack(pady = 10)

#creating a label widget
myl1 = Label(myroot)
myl1.pack(pady=10)

myroot.title('MyScalewidget')
myroot.geometry("300x300+120+120")
myroot.mainloop()
```

Default output:

The default option can be seen in *Figure 4.29*:

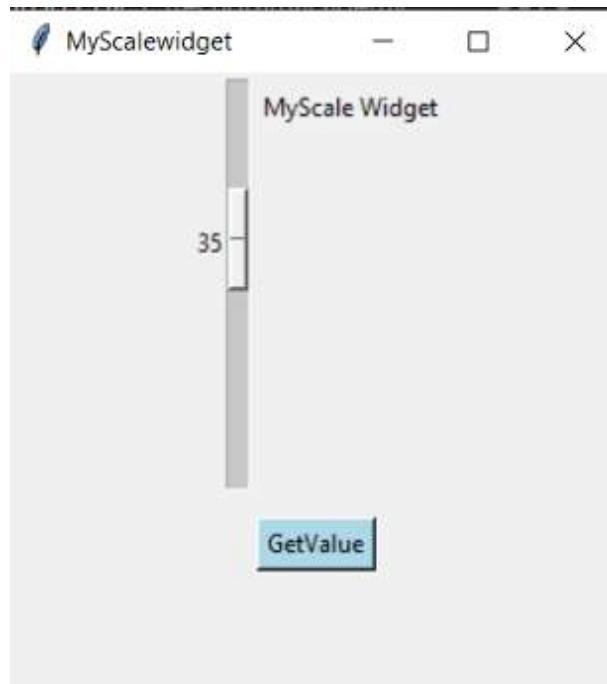


Figure 4.29: Default option

The output when the **GetValue** button is clicked, can be seen in *Figure 4.30*:

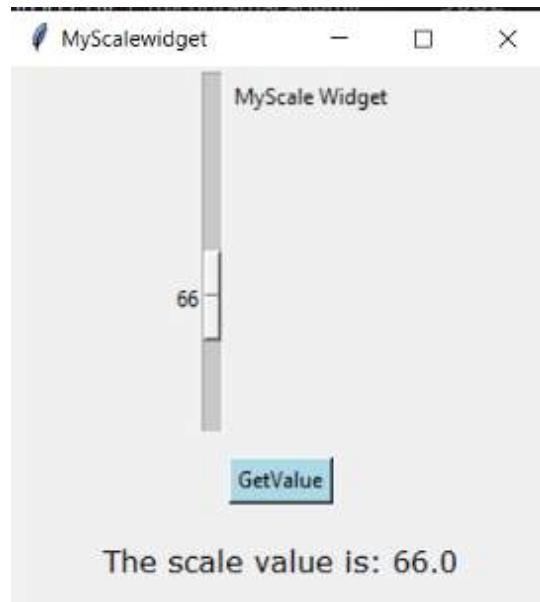


Figure 4.30: Output

Note: The preceding code is covered in Program Name: Chap4_Example18.py

We can change the trough color to any color (here Red) by adding the **troughcolor** option, as shown:

```
mys1 = Scale(myroot, from_=0, to=100, ori-  
ent = VERTICAL, length = 200, width = 10,  
    sliderlength = 50, label = 'MyScale Widget',  
    variable = myv1,troughcolor = 'Red')
```

Refer to *Figure 4.31*:

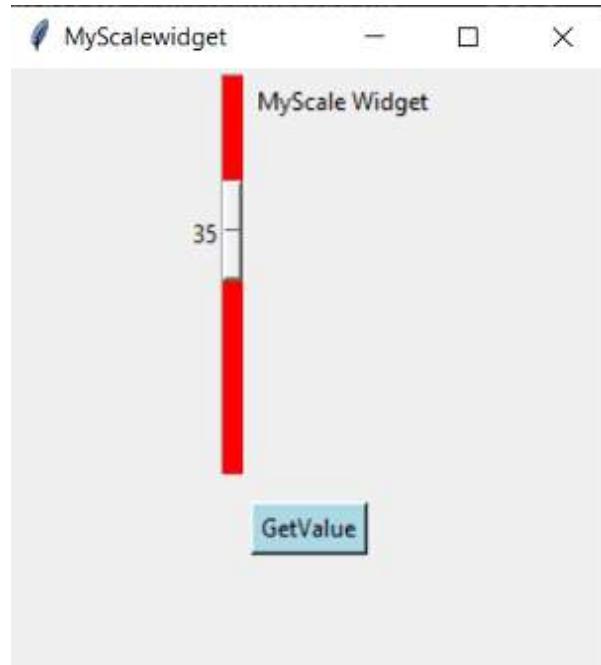


Figure 4.31: Output

We can change the resolution, that is, the slightest change is possible on moving the slider and can provide the scale value at the set intervals, as shown:

```

from tkinter import *

myroot = Tk()

# creating a float variable value holder
myv1 = DoubleVar()

#creation of horizontal slider
mys1 = Scale(myroot, from_=0, to=100, orient = 'horizontal', length = 200, width = 10, sliderlength = 50, label = 'My-Scale Widget', troughcolor = 'Red', resolution = 10, tickinterval = 10)

# setting the scale value to 45
mys1.set(50)
mys1.pack()

myroot.title('MyScalewidget')
myroot.geometry("300x100+120+120")
myroot.mainloop()

```

Output:

Refer to *Figure 4.32*:

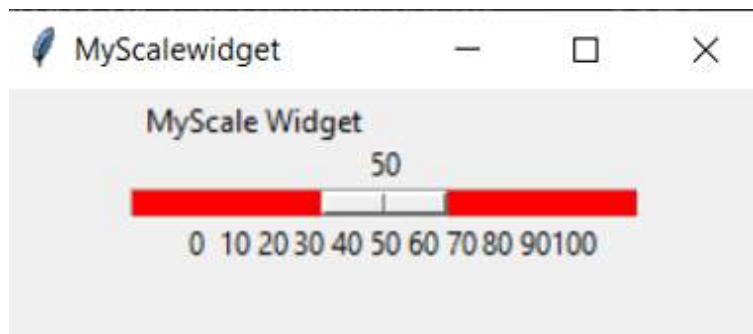


Figure 4.32: Output

Note: The preceding code is covered in Program Name: Chap4_Example19.py

If we set repeatedly = 1000, then button1 will be held down in the trough for 1 sec before the slider starts moving in that direction repeatedly.

tkinter Text widget

This widget will allow options to use text on multiple lines. For a single line, we will be using an **Entry** widget and for multiple lines, we will be using **Text** where we can use elegant structures like marks and tabs for locating different text areas.

The syntax is

```
mytxt1= Text(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bd, bg, cursor, exportselection, fg, font, height, highlightbackground, highlightcolor, highlightthickness, insertborderwidth, insertbackground, insertontime, insertofftime, insertwidth, padx, pady, relief, selectborderwidth, selectbackground, spacing1, spacing2, spacing3, tabs, state, xscrollcommand, yscrollcommand, and wrap.

We have seen most of the options but some undiscussed options are as follows:

- **exportselection**: This option will export the text selected within a text widget in the window manager. If set to 0, it will be suppressed.
- **insertborderwidth**: This option will specify the 3-D border size around the insertion cursor. The default value is 0.
- **insertbackground**: This option will specify the insertion cursor color whose default value is black.
- **insertontime**: This option will specify the time in which the insertion cursor is on during its blink cycle, whose default value are 600

milliseconds.

- **insertofftime:** This option will specify the time in which the insertion cursor is off during its blink cycle, whose default value is 300 milliseconds.
- **insertwidth:** This option will specify the insertion cursor width whose default value is 2 pixels.
- **selectborderwidth:** This option will set the border width which determines the border thickness around selected Text on clicking the mouse and dragging the mouse.
- **spacing1:** This option will specify the amount of vertical space put above each text line. Its default value is 0.
- **spacing2:** This option will specify the amount of extra vertical space added between displayed text lines when a logical line wraps. Its default value is 0.
- **spacing3:** This option will specify how much extra vertical space is added between each text line.
- **tabs:** This option will control the usage of tab characters for positioning the text.
- **state:** This option will represent the DISABLED or NORMAL state of the widget.
- **wrap:** This option will wrap the wider lines into multiple lines. When set to CHAR, will break the line which gets too wider for any character. When set to WORD, will break the line after the last word will fit into the available space.
- **xscrollcommand:** This option when set to the `set()` method of the horizontal scrollbar, the **Text** widget becomes horizontally scrollable.
- **yscrollcommand:** This option when set to the `set()` method of the vertical scrollbar, the Text widget becomes vertically scrollable.

Some of the commonly used methods in the above widget are as follows:

- **delete(startindex[,endindex]):** This method will delete the characters within the given specified range.
- **get(startindex[,endindex]):** This method will fetch the characters within the given specified range.
- **index(index):** This method will return the absolute index of the given specified index.
- **insert(index, string):** This method will insert a string in the given specified index.
- **see(index):** This method will return True if the text specified at the given index is displayed; else will return False.
- **index(mark):** This method will get the index of the specified mark.
- **mark_gravity(mark, gravity):** This method will get the gravity of the mark.
- **mark_names():** This method will fetch the gravity of the given mark.
- **mark_set(mark, index):** This method will specify a new position of the given mark.
- **mark_unset(mark):** This method will remove the provided mark from the above widget.
- **tag_add(tagname, startindex, endindex):** This method will tag the string within the specified range.
- **tag_config():** This method will configure the tag properties.
- **tag_delete(tagname):** This method will delete a given tag.
- **tag_remove(tagname, startindex, endindex):** This method will remove the tag within the specified range.

We shall see some examples of the **Text** widgets. Let us first create a basic example of the **Text** widget, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('400x250')
myroot.title('Textwidget')

# creation of text widget
mytext = Text(myroot, width = 18, height = 10, font = ('Calibiri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

myroot.mainloop() # display window until we press the close button
```

Output:

Refer to the following *Figure 4.33*:

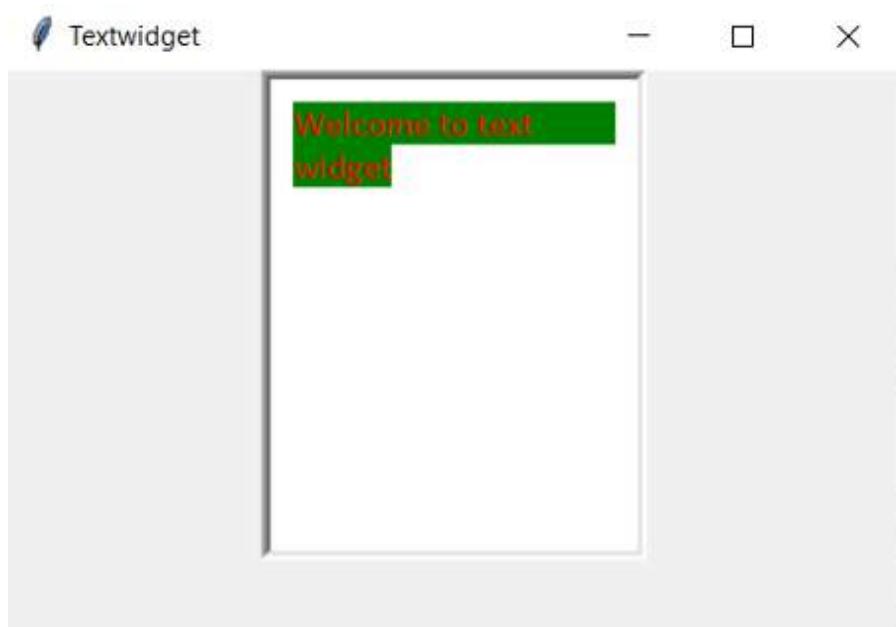


Figure 4.33: Output

Note: The preceding code is covered in Program Name: Chap4_Example20.py

In the above code, we have created a **Text** widget by giving some width and height. We have entered some text and provided the background and foreground color on selection.

There are different **Text** widget indexes that point to some text positions in the **Text** widget which are **line.column**, **line.end**, **insert**, **current**, **end**, **selection**, user-defined tags, windows coordinate ('x', 'y').

We can read the entire contents of the **Text** widget from beginning to end, as shown:

```
from tkinter import * # importing module
from tkinter import messagebox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('400x250')
myroot.title('Textwidget')

# creation of text widget
mytext = Text(myroot, width = 18, height = 10, font = ('Calibri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

#inserting text in the text widget
mytext.insert('1.0', 'Hey Beginners! Welcome for the learning of python text widget. \n This is another line')

# callback function
def myget():
    messagebox.showinfo('Text widget contents are: ',mytext.get('1.0', 'end')) # we are reading the entire contents of the text widget and displaying

# creation of button widget
mybtn1 = Button(myroot, text = 'Read', command = myget)
mybtn1.pack()

myroot.mainloop() # display window until we press the close button
```

Output:

Refer to *Figure 4.34*:

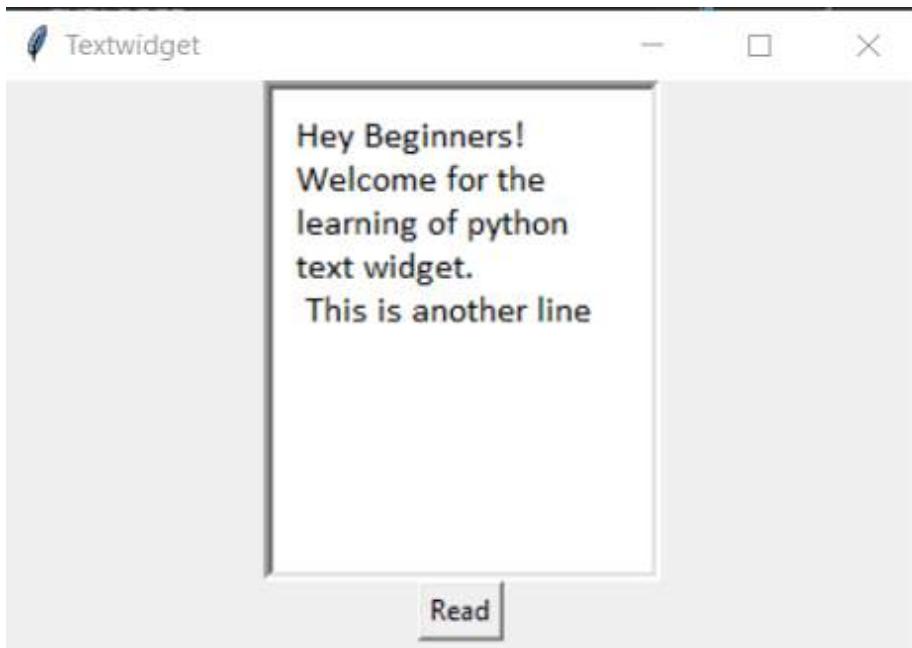


Figure 4.34: Output

The output when the **Read** button is clicked, can be seen in the following *Figure 4.35*:

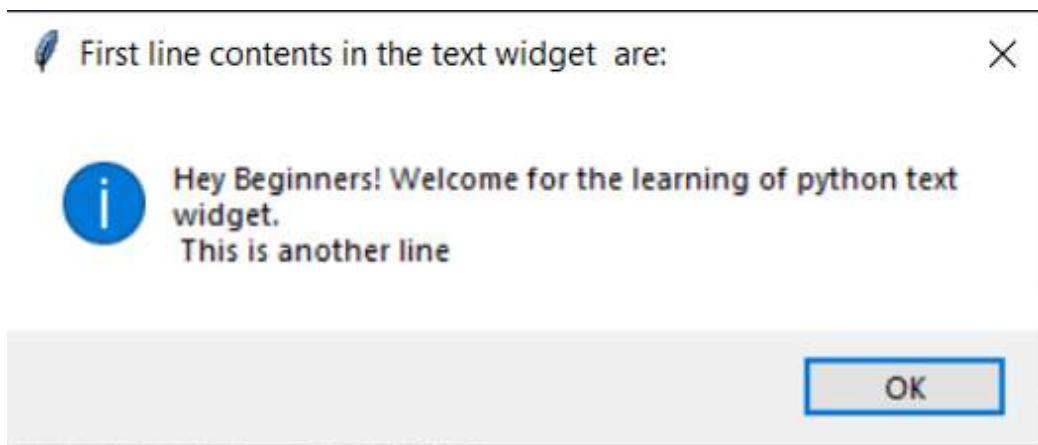


Figure 4.35: Output

Note: The preceding code is covered in Program Name: Chap4_Example21.py

In the same example, we can get the contents of the first line by modifying the code inside the callback function, as shown:

```
# callback function
def myget():
    messagebox.
    showinfo('First line contents in the text widget are: ',mytext.
get('1.0', '1.end')) # we are reading the contents of first line only
```

The output when the **Read** button is clicked, can be seen in *Figure 4.36*:

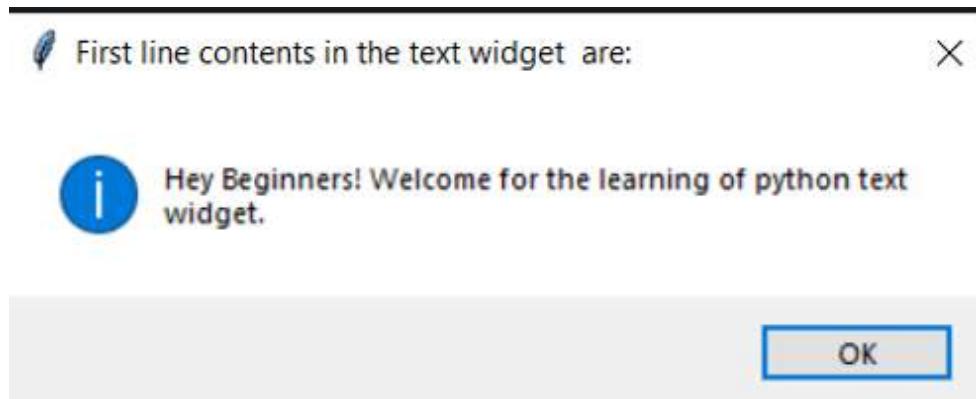


Figure 4.36: Output when Read Button is clicked

We can also insert text in the third line into the text widget by adding the following lines in the code:

```
# inserting text in the third line
mytext.insert('1.0 + 2 lines', '\nThis is 3rd line')
```

Output:

Refer to *Figure 4.37*:

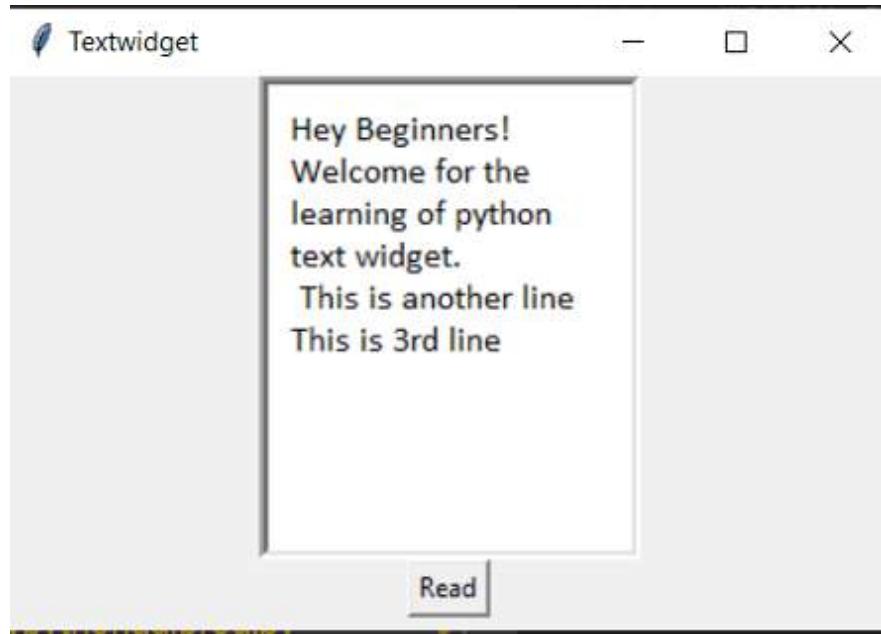


Figure 4.37: Output when text is inserted in the third line

We can delete a single character and an entire line in the text widget as follows:

```
from tkinter import * # importing module
from tkinter import messagebox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('400x350')
myroot.title('Textwidget')

# creation of text widget
mytext = Text(myroot, width = 18, height = 10, font = ('Calibri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

#inserting text in the text widget
mytext.insert('1.0', 'Hey Beginners! Welcome for the learning of python text widget. \n This is another line')

# inserting text in the third line
mytext.insert('1.0 + 2 lines', '\nThis is 3rd line')

# callback function
def myget():
    messagebox.showinfo('First line contents in the text wid-
get are: ',mytext.get('1.0', '1.end')) # we are reading the con-
tents of first line only
```

```
# creation of button widget
mybtn1 = Button(myroot, text = 'Read', command = myget)
mybtn1.pack()

def mydelete():
    mytext.delete('1.0')

# creation of Delete button for single characterwidget
mybtn2 = Button(myroot, text = 'DeleteSingleCharacter', com-
mand = mydelete)
mybtn2.pack(pady = 10)

def mydelete_entireline():
    mytext.delete('1.0','1.0 lineend')

# creation of Delete button for entire line widget
mybtn2 = Button(myroot, text = 'DeleteEntireLine', com-
mand = mydelete_entireline)
mybtn2.pack(pady = 10)

myroot.mainloop() # display window until we press the close button
```

Output:

Refer to *Figure 4.38*:

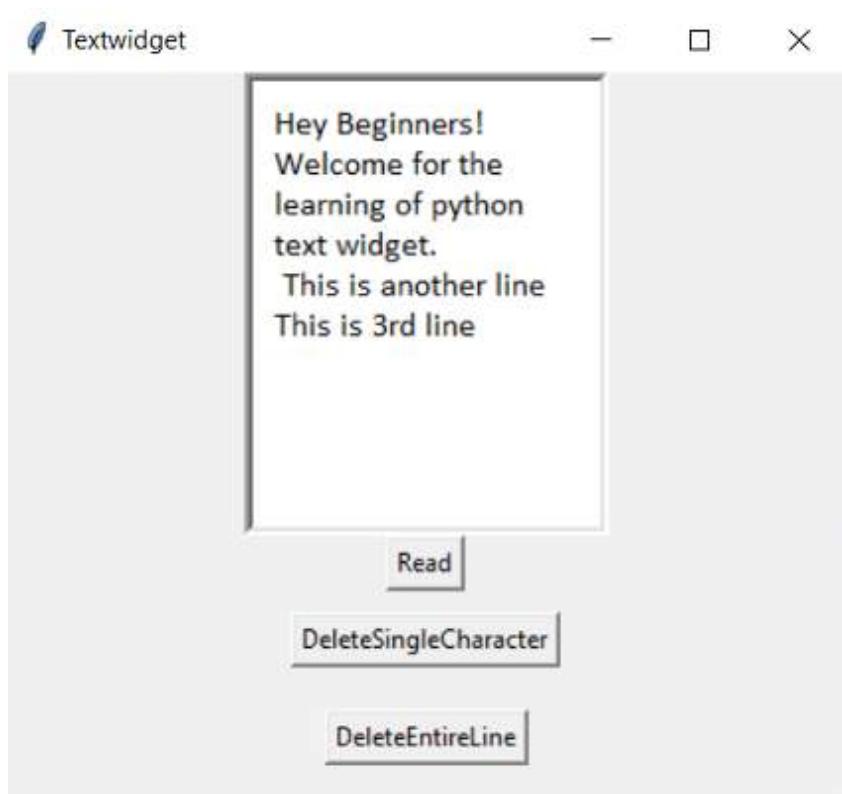


Figure 4.38: Output

The output when the **DeleteSingleCharacter** button is clicked, can be seen in [*Figure 4.39*](#):



Figure 4.39: Output when DeleteSingleCharacter button is clicked

The output when the **DeleteEntireLine** button is clicked, can be seen in [Figure 4.40](#):



Figure 4.40: Output when *DeleteEntireLine* button is clicked (This will delete the first line)

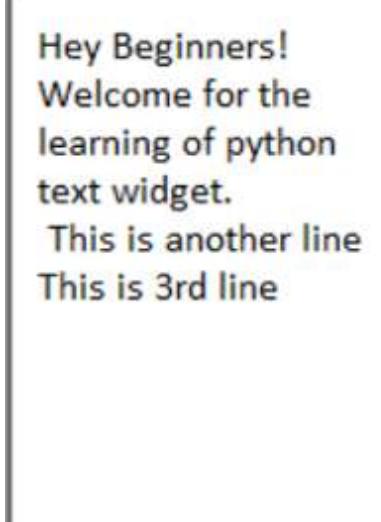
Note: The preceding code is covered in Program Name: **Chap4_Example22.py**

We can replace the text in the **Text** widget with some new text. Here, we are replacing the entire first line with a new text by adding the following line, as shown:

```
mytext.replace('1.0','1.0 lineend', 'This is first line')
```

Output:

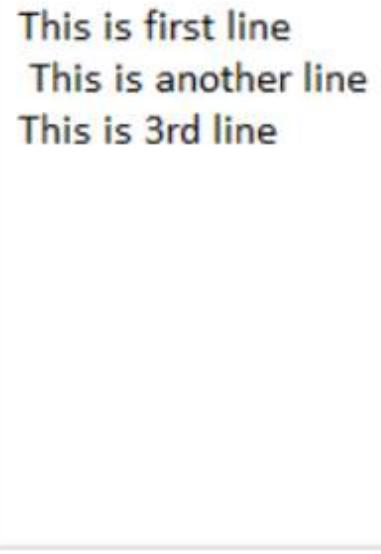
The output before replacement can be seen in the following *Figure 4.41*:



```
Hey Beginners!
Welcome for the
learning of python
text widget.
This is another line
This is 3rd line
```

Figure 4.41: Output before replacement of first line

The output after replacement can be seen in the following [Figure 4.42](#):



```
This is first line
This is another line
This is 3rd line
```

Figure 4.42: Output after replacement of first line

We can disable the state of the **Text** widget by using the **state** option:

```
mytext.config(state = 'disabled')
```

Refer to [Figure 4.43](#):



Figure 4.43: Output for disabling the Text widget state

Even if we will try to press any delete buttons, the text in the **Text** widget will not be deleted. Hence, to perform any operation, we will revert back the state to normal.

```
mytext.config(state = 'normal')
```

Now, we shall see how we can identify and name sections of the text with tags and marks. Tags shall describe a range of collections of characters and marks specify a specific location between 2 characters within the text widget. We will use the *tags* and *marks* to change properties such as the *font* and *color* for sections of text and control where to insert and delete text.

Now, to add a tag to the **Text** widget, we can use the **tag_add** method, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x300')
myroot.title('Textwidget')

mytext = Text(myroot, width = 18, height = 10, font = ('Calibiri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

mytext.insert('1.0', 'This is 1st line')
mytext.insert('1.0 + 1 line', '\nThis is 2nd line')
mytext.insert('1.0 + 2 lines', '\nThis is 3rd line')

# 1st par: Name of the tag which will be created as a string
# 2nd par: start
# 3rd par: end
mytext.tag_add('mytag1','1.0','1.0 wordend')

# Now, we can configure properties about the tag using tag_configure
mytext.tag_configure('mytag1', background = 'Pink')

myroot.mainloop()
```

Output:

Refer to *Figure 4.44*:

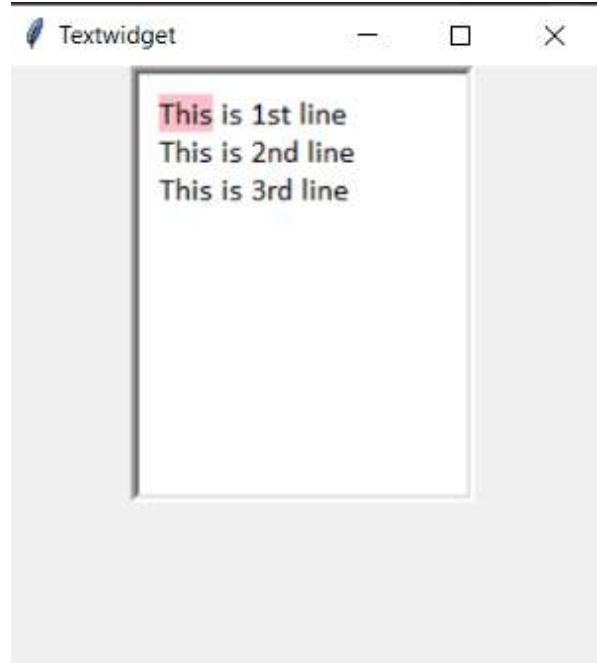


Figure 4.44: Output

Note: The preceding code is covered in Program Name: Chap4_Example23.py

So, we can see from the above code that the word *This* is highlighted, which is the first word in the first line.

We can find out the characters included in a tag by adding the following line after the `tag_configure()` method in the code:

```
print(mytext.tag_ranges('mytag1'))
```

Output:

Refer to the following *Figure 4.45*:

```
(<textindex object: '1.0'>, <textindex object: '1.4'>)
```

Figure 4.45: Output for characters search included in a tag

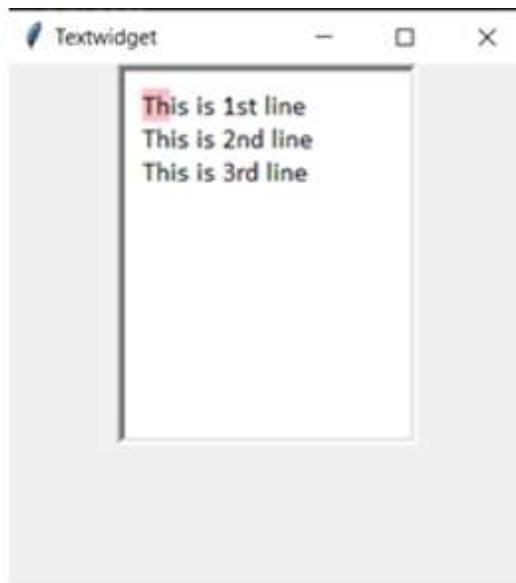
Here, `tag_ranges()` will return the start and end locations of the sections covered by the tag.

We can remove a tag from the specified range. Just add the following lines of code after the `tag_ranges()` method.

```
mytext.tag_remove('mytag1', '1.2','1.4')
print(mytext.tag_ranges('mytag1'))
```

Output:

Refer to *Figure 4.46*:



```
(<textindex object: '1.0'>, <textindex object: '1.4'>)
(<textindex object: '1.0'>, <textindex object: '1.2'>)
[]
```

Figure 4.46: Output for tag removing from the specified range

We can use the tag as an index by using `replace()` method. Just add the following lines of code:

```
mytext.replace('mytag1.first','mytag1.last', 'Hereit ')
mytext.tag_add('mytag1','1.0','1.0 lineend')
mytext.tag_configure('mytag1', background = 'Blue')
```

Refer to the following *Figure 4.47*:

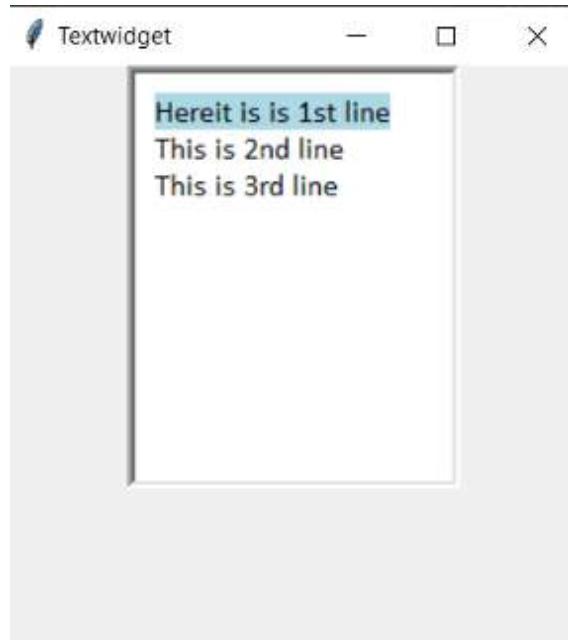


Figure 4.47: Output with usage of replace method

We can delete a tag by using the following line.

```
mytext.tag_delete('mytag1')
```

Till now, we have discussed tags and now we will discuss about marks. To get the list of marks present in the above widget, we will use the following lines of code:

```

from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x300')
myroot.title('Textwidget')

# creation of text widget
mytext = Text(myroot, width = 18, height = 10, font = ('Calibri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

mytext.insert('1.0', 'This is 1st line')
mytext.insert('1.0 + 1 line', '\nThis is 2nd line')
mytext.insert('1.0 + 2 lines', '\nThis is 3rd line')

print(mytext.mark_names()) # by default there are 2 marks that tk automatically keeps track of.

myroot.mainloop()

```

Output:

Refer to *Figure 4.48*:

```
$ python mypythonguiproj.py
('insert', 'current')
```

Figure 4.48: Output

Note: The preceding code is covered in Program Name: Chap4_Example24.py

We can see that there are 2 automatically tracked text marks that are inserted and current. The first mark insert is the insertion cursor current index and another current mark is the automatically tracked mark and will specify the index which is currently under the mouse.

Now, we shall see the usage of an automatically tracked insert mark as the index for an insert method, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x330')
myroot.title('Textwidget')

# creation of text widget
mytext = Text(myroot, width = 15, height = 10, font = ('Calibri',12), wrap = WORD, padx = 10, pady = 10, bd = 4, selectbackground = 'Green', selectforeground = 'Red')
mytext.pack()

mytext.insert('1.0', 'This is 1st line')
mytext.insert('1.0 + 1 line', '\nThis is 2nd line')

def myinsert_mark():
    mytext.insert('insert','@') # will insert '@' at the position of the insert marker

mybtn2 = Button(myroot, text = 'InsertMark', command = myinsert_mark)
mybtn2.pack(pady = 10)

myroot.mainloop()
```

Output:

Refer to *Figure 4.49*:



Figure 4.49: Output

We can see that the cursor is placed before the line.

The output, when insert mark button, is clicked, can be seen in the following [**Figure 4.50:**](#)



Figure 4.50: Output

Note: The preceding code is covered in Program Name: Chap4_Example25.py

When the **InsertMark** button is clicked, '@' symbol is inserted where the cursor position was placed.

We can also create and modify the location of the mark using the **mark_set()** method. We can mark text in the **tkinter** text widget, as shown:

```

from tkinter import *

myroot=Tk()

def myclick():
    mytext.insert('insert','<>')
    mytext.mark_names() # all the mark names are returned
    mytext.mark_
set('insert',END) # a new position is informed of the given mark
    mytext.mark_
gravity('insert',RIGHT) # changing the gravity of mark to right
mybtn1=Button(myroot,text="Myclick",command=myclick)
mybtn1.pack()

mytext=Text(myroot , width = 55, height = 10)
mytext.pack()

myroot.mainloop()

```

Output:

The output can be seen in the following *Figure 4.51*:



Figure 4.51: Output

Note: The preceding code is covered in Program Name: Chap4_Example26.py

We have clicked **Myclick** initially 4 times and entered the text **hellothere**. Then, we again click the button 3 times and place the cursor in the position after the text **hellothere**. Now when **Myclick** button is clicked again, then '**<>**' will be inserted in the text widget and the cursor will be placed after this, as shown in *Figure 4.52*:

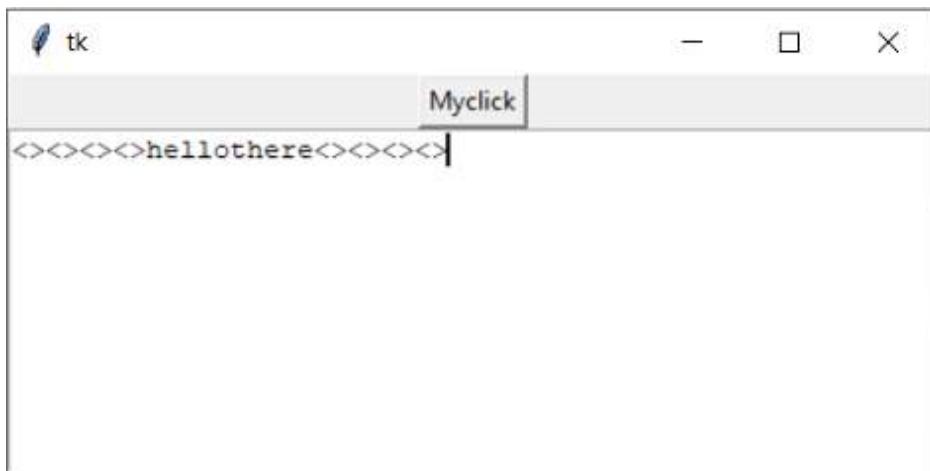


Figure 4.52: Output display on pressing Myclick button 4th time after text hellothere

However, if we comment on the three lines as shown:

```
def myclick():
    mytext.insert('insert','<>')
    # mytext.mark_names() # all the mark names are returned
    # mytext.mark_
    set('insert',END) # a new position is informed of the given mark
    # mytext.mark_
    gravity('insert',RIGHT) # changing the gravity of mark to right
```

If we run the same program now and get into the same output position, we have the following *Figure 4.53*:



Figure 4.53: Output display

Now, if we click the **Myclick** button again, see the position of the cursor as in [Figure 4.54](#):

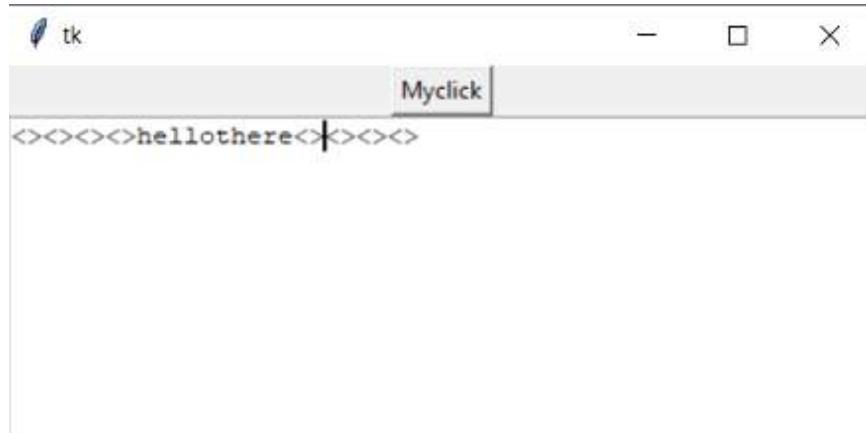


Figure 4.54: Output display after commenting the 3 lines and pressing Myclick button

We can also insert an image in the text widget, as shown:

```
from tkinter import *

myroot=Tk()

def myclick():
    mytext.insert('insert',"<>")

mybtn1=Button(myroot,text="Myclick",command=myclick)
mybtn1.pack()

mytext=Text(myroot, width = 55, height = 25)
mytext.pack()

def insertimage():
    mytext.image_create('insert',image = myimage1)

myimage1 = PhotoImage(file = 'butterfly1.gif')
mybtn2=Button(myroot,text="CreateImage",command = insertimage)
mybtn2.pack(pady = 10)

myroot.mainloop()
```

Output:

Refer to *Figure 4.55*:

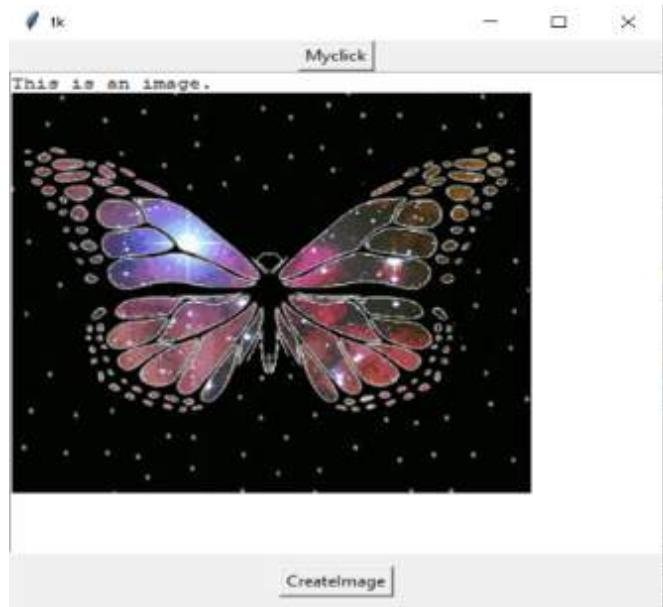


Figure 4.55: Output

Note: The preceding code is covered in Program Name: Chap4_Example27.py

We have already seen using the scrollbar with the **Text** widget earlier in this chapter. So, we will not discuss that here.

tkinter Combobox Widget

This widget is a combination of a drop-down menu and an **Entry** widget. Here, the user can view the usual text entry area with a downward pointing arrow. A drop-down menu appears when the user clicks on the arrow displaying all the choices and will replace the current entry contents if clicked on one.

The syntax is as follows:

```
mycmb1= Combobox(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are cursor, exportselection, height, justify, style,

postcommand, takefocus, validate, validatecommand, textvariable, width, values, and xscrollcommand.

We have seen most of the options but some undiscussed options are as follows:

- **exportselection**: Whenever the text is selected within an entry widget and if exportselection is set to 0, the automatic export to the clipboard is restricted.
- **postcommand**: When the user clicks on the down arrow, this option can be set to any of the functions.
- **values**: This option specifies the choices as a sequence of strings which will appear in the drop-down menu.

Some of the methods used in the above widget are as follows:

- **current([index])**: This method when passing the index of the element as an argument will select one of the elements of the values option. If an argument is not supplied, the value returned will be the index of the current Entry text in the values list.
- **set(value)**: This method can set the current text in the widget to value.

Let us see an example for better understanding:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
myroot.title('Comboboxcreation')

# creating a list of values
myl2 = list(range(1,25))

#combobox object creation
mycombo = Combobox(myroot, values = myl2 , width = 15) # , height = 2 : only 2 items
mycombo.pack(padx = 50, pady = 10)

myroot.mainloop() # display window until we press the close button
```

Output:

The output can be seen in the following *Figure 4.56*:

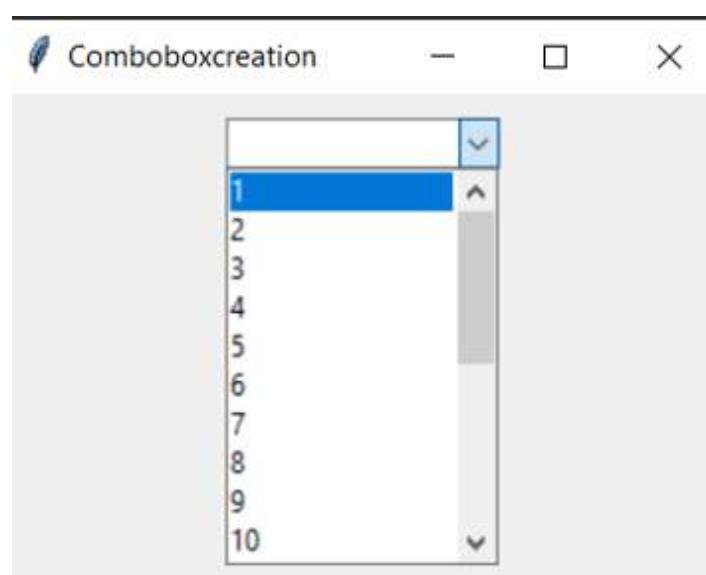


Figure 4.56: Output

Note: The preceding code is covered in Program Name: Chap4_Example28.py

In this code, we are passing an integer list in the values option of the **combobox** object. The user can choose any of the values from the available drop-down menu. The user clicking on any of the options will be available in the **Entry** widget, as shown in *Figure 4.57*:

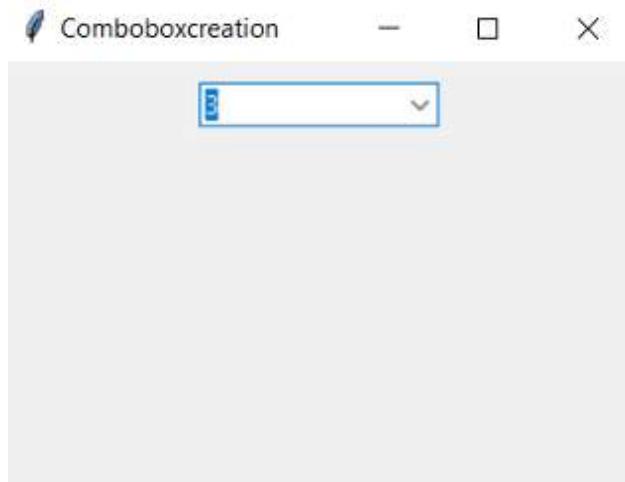


Figure 4.57: Output display on selecting any particular value from the combobox widget

We can also add a string list in the values option of the **combobox** object, as shown:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
```

```

myroot.title('Comboboxcreation')

# creating a list of values
myl1 = ['Hindi', 'English', 'Telugu', 'Bengali']

#combobox object creation
mycombo = Combobox(myroot, val-
ues = myl1 , height = 2) # , height = 2 : only 2 items and de-
fault width is 20
mycombo.pack()

myroot.mainloop() # display window until we press the close button

```

Output:

Refer to *Figure 4.58*:

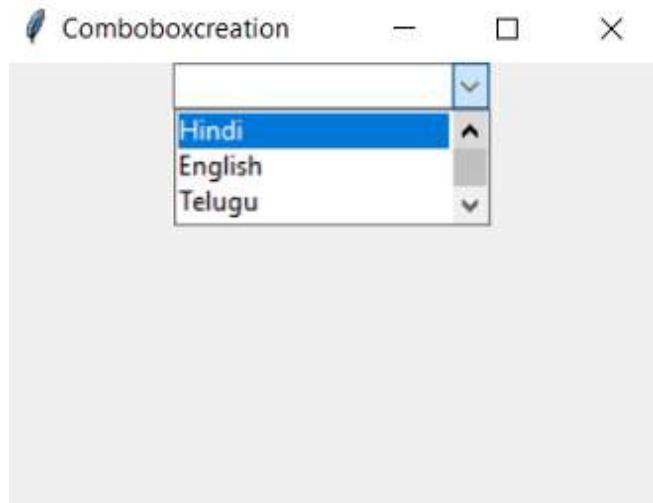


Figure 4.58: Output

Note: The preceding code is covered in Program Name: Chap4_Example29.py

We can also display the text in the **combobox** widget on the selection of the drop-down menu, as shown:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
myroot.title('Comboboxcreation')

myval = StringVar()

def mydisplay():
    myval = mycombo.get()
    print(myval)

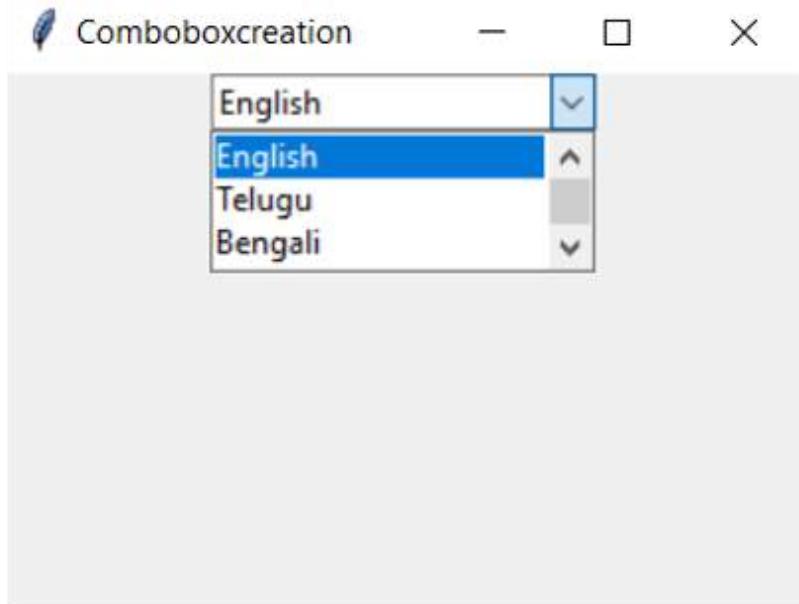
# creating a list of values
myl1 = ['Hindi','English','Telugu','Bengali']
myval.set('English')

mycombo = Combobox(myroot, values = myl1 , height = 2, textvariable = myval ,postcommand = mydisplay) # , height = 2 : only 2 items and default width is 20
mycombo.pack()

myroot.mainloop() # display window until we press the close button
```

Output when the drop-down menu is selected:

Refer to *Figure 4.59*:



```
$ python mypythonguiprog.py
English
█
```

Figure 4.59: Output

Note: The preceding code is covered in Program Name: Chap4_Example30.py

In the above code, by default, we have set it to English in the above widget. The user can select any of the options from the above widget. Whenever the user clicks on the down arrow, a callback function will be invoked which will display the text present in the console.

Moreover, we could have used the current method and passed the index of the element to be displayed instead of `myval.set('English')`, as shown:

```
mycombo.current(1)
```

So, the `get()` method will return the element itself whereas `current()` will get the index of the currently selected element.

However, what if we want to display the output to the console after the element is selected from the combobox list? In such a case, the virtual event <<ComboboxSelected>> is bonded with the callback function, as shown:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
myroot.title('Comboboxcreation')

myl2 = Label(myroot, text = 'Choose your mother tongue')
myl2.pack(pady = 10)

def mydisplay(myevent):
    print(mycombo.get())

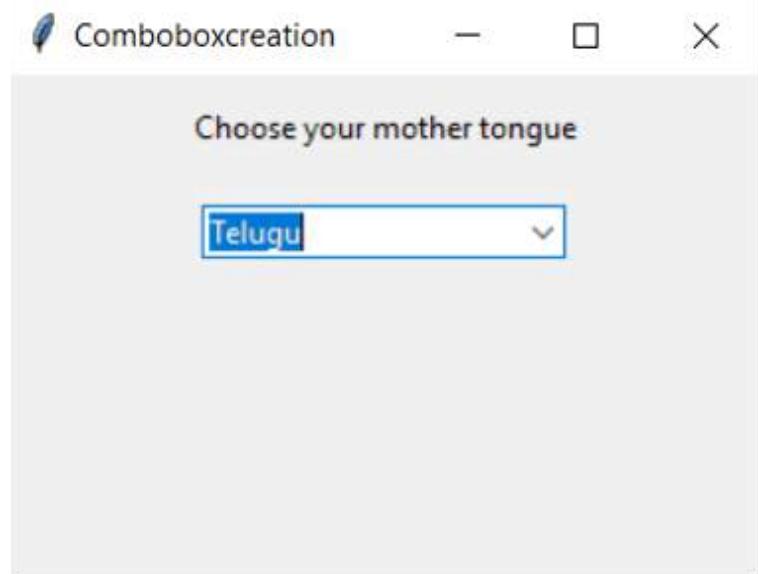
# creating a list of values
myl1 = ['Hindi', 'English', 'Telugu', 'Bengali']

#combobox object creation
mycombo = Combobox(myroot, values = myl1 , height = 2) # , height = 2 : only 2 items and default width is 20
mycombo.pack(pady = 10)
mycombo.current(1)
mycombo.bind("<<ComboboxSelected>>", mydisplay)

myroot.mainloop() # display window until we press the close button
```

Output after the element is selected from the drop-down menu:

Refer to *Figure 4.60*:



```
$ python mypythonguiprog.py  
Telugu  
[]
```

Figure 4.60: Output

Note: The preceding code is covered in Program Name: Chap4_Example31.py

We can set fonts for both combobox and its listbox. If unspecified, then the text in the combobox list will still use the system default font but not the font specified to the combobox, as shown:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
myroot.title('Comboboxcreation')

# creating a list of values
mylist1 = ['Apple','Litchi','Mango','Pomengranate']

# assigning font tuple
myfont = ("Times New Roman", 14, "italic")

#assigning label
myl1 = Label(myroot, text = 'Choose from your favorite fruit')
myl1.pack(pady = 10)

#combobox object creation
mycombo = Combobox(myroot, values = mylist1 , height = 2, font = myfont) # , height = 2 : only 2 items and default width is 20
mycombo.pack(pady = 10)
mycombo.current(1)

# specifying font of the list box of Combobox
myroot.option_add('*TCombobox*Listbox.font', myfont)

myroot.mainloop() # display window until we press the close button
```

Output:

Refer to *Figure 4.61*:

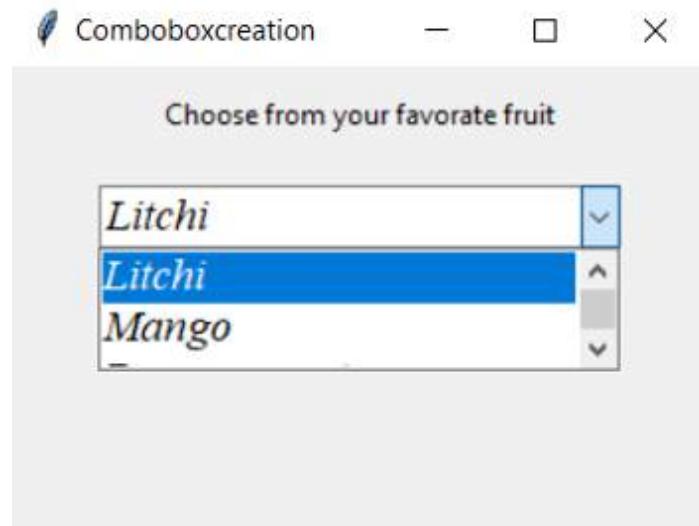


Figure 4.61: Output

Note: The preceding code is covered in Program Name: Chap4_Example32.py

We can dynamically change the value of the comobox list values, as shown:

```
from tkinter import * # importing module
from tkinter.ttk import Combobox

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('300x200')
```

```

myroot.title('Comboboxcreation')

myl2 = Label(myroot, text = 'Choose your mother tongue')
myl2.pack(pady = 10)

def mydisplay():
    mycombo['values'] = ['Punjabi', 'Tamil', 'Spanish','Kannada']
    mycombo.set('')

# creating a list of values
myl1 = ['Hindi','English','Telugu','Bengali']

#combobox object creation
mycombo = Combobox(myroot, values = myl1 , height = 2, post-
command = mydisplay) # , height = 2 : only 2 items and de-
fault width is 20
mycombo.pack(pady = 10)
mycombo.current(1)

myroot.mainloop() # display window until we press the close button

```

Output:

The output can be seen in *Figure 4.62*:

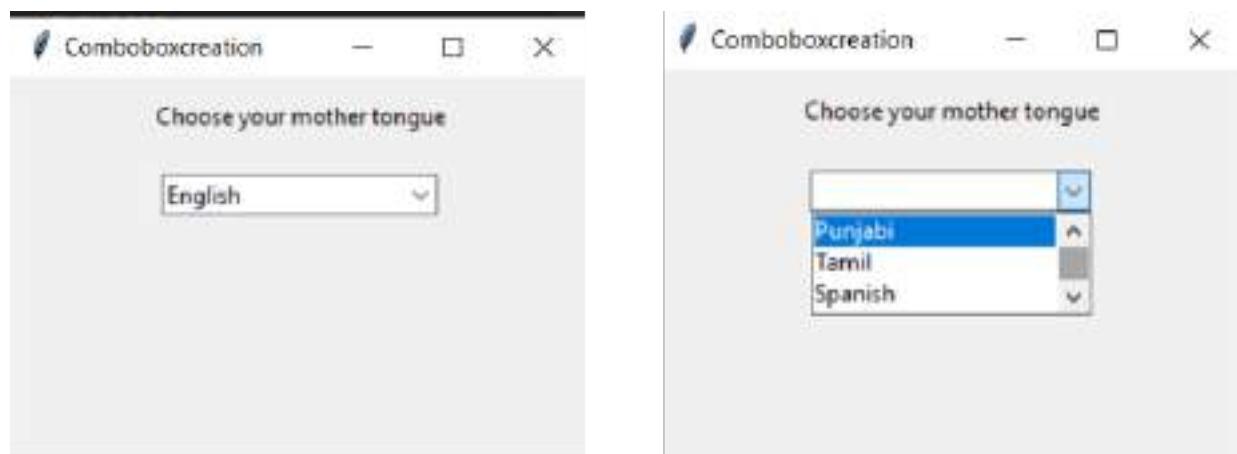


Figure 4.62: Output

Note: The preceding code is covered in Program Name: Chap4_Example33.py

Conclusion

This chapter has discussed a variety of tkinter widgets, including the Entry, Scrollbar, Spinbox, Scale, Text, and Combobox. The chapter offers thorough explanations, examples, and alternatives for utilizing these widgets to build straightforward GUI applications. The concept of validation in the Entry widget, the ability to scroll in the Scrollbar widget, the ability to select values from a range in the Spinbox widget, the implementation of a graphical slider in the Scale widget, the ability to insert multiple text fields in the Text widget, and the use of the Combobox widget and its applications are all neatly and clearly covered.

Points to remember

- Entry widget can be used for gathering user input, can be customized using various controls viz width, font, and validation, and can be utilized for creating a simple data entry application; the **get()** method can be retrieved to get the input.
- Scrollbar widgets are used to provide widgets that are longer than their visible size, a scrolling feature. It is compatible with a variety of widgets, including Listbox, Text, Canvas and so on. We can position the scrollbar either vertically or horizontally. The scrollbar should be attached to the widget that needs to be scrolled using the **command** option.
- Using **Text** widget, we can display and edit the text in multiple lines. We can customize using various options such as color, font, wrap and so on. It can be used to build applications such as text editors or document viewers. The **get()** method can be used to get the text from the widget.
- The **SpinBox** widget is used to select a value from a fixed range of values. We can customize using various options such as color, font,

range and so on. It can be used to create a simple data selection application. The selected value can be retrieved using `get()` method.

- The **Scale** widget is used to select from a range of numbers by providing a graphical slider object and moving through a slider. We can customize using various options such as color, font and so on. It can be used to create an application which requires value to be selected within a certain range. The selected value can be retrieved using `get()` method.
- Combination of a drop-down menu and an **Entry** widget is a **Combobox** widget. We can customize using various options such as color, font, values and so on. It can be used to create an application which requires user to select one option from a list of options. The selected option can be retrieved using `get()` method.

Questions

1. Explain the tkinter Entry widget in detail.
2. Which widget accepts single-line text strings from the user? Explain in detail with a suitable example.
3. How is the validation on an entry widget done? Explain the process in detail.
4. Explain the tkinter Scrollbar widget and its syntax.
5. Explain tkinter Spinbox Widget and its usage in GUI designing.
6. How does the user choose some fixed range of values? Explain the widget used for this purpose in detail.
7. Which widget is alternatively used in place of the entry widget? Explain with suitable justification.
8. Explain tkinter Scale Widget and its syntax.
9. Which widget allows one to select from a range of numbers by providing a graphical slider object and moving through a slider? Explain this widget with suitable justification and example.

10. Explain the tkinter Text Widget in detail.
11. Explain tkinter Combobox Widget in detail with a sample program.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 5

Getting Insights of Display Widgets in tkinter

Introduction

The Label, Message, and MessageBox widgets in Tkinter must be learned in order to create Python **Graphical User Interfaces (GUI)**, which are efficient and easy to use. We will be able to display text, images, and dialog boxes to the user, thanks to these widgets which are necessary for creating an effective and user-friendly GUI. We can provide data to the user in a clear and organized manner, using the Label and Message widgets. We could use a Label widget to show a user's name or a Message widget to show a list of items, for instance. We must use the MessageBox widget to handle errors in our application. We are able to ask the user for input, such as whether to continue or cancel an operation, and display error warnings to them. These widgets offer a variety of customization options, including the ability to change the text, picture, and dialog box size, colour, and font. For an application's user interface to be unique and different from others, customization is essential.

Structure

In this chapter, we will discuss the following topics:

- tkinter Label Widget
- tkinter Message Widget
- tkinter MessageBox Widget

Objectives

By the end of this chapter, the reader will learn about the creation of a simple GUI app using the tkinter Label widget which depicts the ways of displaying a text or image on a window form. We shall also view a display of prompt unedited text messages to the user, with the tkinter Message widget. Moreover, we will look into multiple message boxes such as information, warning, error, and so on, in a Python application by using the tkinter MessageBox widget.

tkinter Label Widget

It is a standard tkinter widget where a text or an image can be displayed on the screen. The text can be underlined, can be displayed in a single font and the text may be spanned across multiple lines. It uses double buffering so that the contents may be updated at any time without display of any flickering. In this widget, one or more lines of text can be displayed which cannot be modified by the user.

The syntax is as follows:

```
myl1= Label(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the list of options that can be used as key-value pairs and are separated by commas, are anchor (space between are and anchor), bg, bitmap, bd, cursor, font, height, fg, justify, image, padx, pady, text, textvariable, relief, width, wraplength, underline.

We have seen most of the options but some undiscussed options are as follows:

- **text:** This option will display the text on the label. The '\n' will force a line break. So, one or more lines of text in the label can be displayed. This option is ignored when the image or bitmap options are used.

- **textvariable**: This option will associate a tkinter variable (generally the StringVar) with the label. The label text is updated if the variable is changed.
- **image**: This option will display the image in the label and the value should be a BitmapImage or a PhotoImage. This option takes precedence over the bitmap and text options.
- **justify**: This option will define the alignment of multiple lines of text with respect to each other. The default is CENTER and the other is LEFT or RIGHT.

Let us make a basic label widget in the parent window:

```
from tkinter import *

myroot = Tk() # creating an object of Tk class -- object of window

myroot.maxsize(300,300) # maximum size of window- It can be smaller like anything byt maximum up to 300 only
myroot.resizable(0,0) # window size is fixed. cannot be larger or smaller.

mytk_label = Label(myroot,text = 'Python\nis\nawesome', font = ('Calibri',15),bg = 'Yellow',fg = 'Black',
                   width = '15', height = '3')
mytk_label.pack()

myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.1*:



Figure 5.1: Output

Note: The preceding code is covered in Program Name: Chap5_Example1.py

In the above code, we passed the window object into the Label constructor and set the text property, which becomes the text Label when displayed. Other options such as font, bg, fg, width, and height for the font color, size, desired width, and height are defined and used to pack geometry manager for organizing the label widget in the block. Here, we have displayed multiple lines of text in a label.

We can set the look of the border of a label by using the borderwidth and relief options as shown:

```
from tkinter import *
myroot = Tk()
```

```
myl1_label = Label(myroot,text = 'Hey! I', bd = 2, relief = 'solid',font = ('Calibri',15))
myl1_label.pack()

myl2_label = Label(myroot,text = 'love', bd = 5, relief = 'sunken',font = ('Calibri',15), padx = 10,pady = 10)
myl2_label.pack(padx = 10, pady = 10)

myl3_label = Label(myroot,text = 'python', bd = 5, relief = 'raised',font = ('Calibri',15), padx = 10,pady = 10)
myl3_label.pack(padx = 10, pady = 10)

myl4_label = Label(myroot,text = 'to', bd = 5, relief = 'groove',font = ('Calibri',15), padx = 10,pady = 10)
myl4_label.pack(padx = 10, pady = 10)

myl5_label = Label(myroot,text = 'read', bd = 5, relief = 'groove',font = ('Calibri',15), padx = 10,pady = 10)
myl5_label.pack(padx = 10, pady = 10)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.2*:



Figure 5.2: Output

Note: The preceding code is covered in Program Name: [Chap5_Example2.py](#)

We can position text within a label widget as shown:

```

from tkinter import *

class MyLabelPosition(Tk):
    def __init__(self):
        super().__init__()
        self.title('Position Text within a label')
        self.myl2= Label(self, text = 'Hello\\nThere',bd = 4, relief = 'groove', font = 'Times 32', width = 10,
                         height = 4, anchor = SW)
        self.myl2.pack()

if __name__ == "__main__":
    myroot = MyLabelPosition()
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.3*:



Figure 5.3: Output

Note: The preceding code is covered in Program Name: Chap5_Example3.py

In the above case, we are positioning the text within the label widget in South West direction.

Now, we will see how to pad space around the text of a label:

```

from tkinter import *

class MyPadSpace(Tk):
    def __init__(self):
        super().__init__()
        self.title('Pad space around the text')
        myl1 = Label(self, text = 'Python')
        myl1.pack()
        myl2= Label(self, text = 'Stay\Safe',bd = 4, relief = 'groove',font = ('Verdana',12))
        myl2.pack()
        myl3 = Label(self, text = 'Python')
        myl3.pack()
        myl4= Label(self,text = 'Stay\Safe',bd = 4, relief = 'groove',font = ('Verdana',12),padx = 20)
        myl4.pack()
        myl5 = Label(self, text = 'Python')
        myl5.pack()
        myl6= Label(self,text = 'Stay\Safe',bd = 4, relief = 'groove',font = ('Verdana',12),pady = 10)
        myl6.pack()
        myl7 = Label(self, text = 'Python')
        myl7.pack()
        myl8= Label(self,text = 'Stay\Safe',bd = 4, relief = 'groove',font = ('Verdana',12),padx = 10, pady = 10)
        myl8.pack()

if __name__ == "__main__":
    myroot = MyPadSpace()
    myroot.geometry('350x300')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.4*:



Figure 5.4: Output

Note: The preceding code is covered in Program Name: Chap5_Example4.py

Now, we will see how to justify text in a label:

```

from tkinter import *

class MyJustify(Tk):
    def __init__(self):
        super().__init__()
        self.title('Jusify in label')
        myl1 = Label(self, text = 'Python')
        myl1.pack()
        myl2= Label(self, text = 'Hello\nThere There\
nThere There There',bd = 2, relief = 'solid',font = ('Helvetica',10))
            # default justify is CENTER
        myl2.pack()
        myl3 = Label(self, text = 'Python')
        myl3.pack()
        myl4= Label(self,text = 'Hello\nThere There\
nThere There There',bd = 2, relief = 'solid',font = ('Helvetica',10),
                    justify = LEFT)
        myl4.pack()
        myl5= Label(self,text = 'Hello\nThere There\
nThere There There',bd = 2, relief = 'solid',font = ('Helvetica',10),
                    justify = RIGHT)
        myl5.pack()

if __name__ == "__main__":
    myroot = MyJustify()
    myroot.geometry('350x300')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.5*:

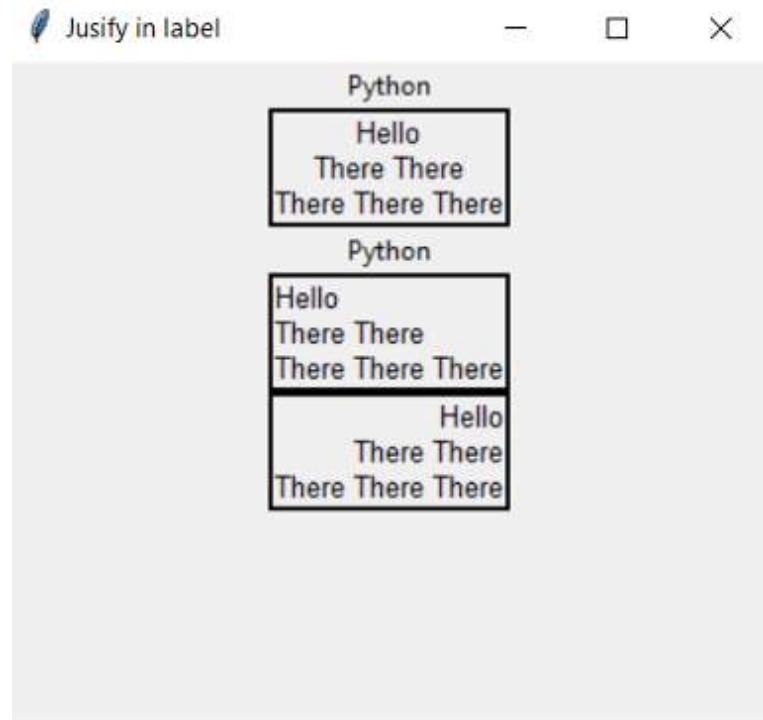


Figure 5.5: Output

Note: The preceding code is covered in Program Name: Chap5_Example5.py

Moreover, we can anchor and justify text simultaneously in a label, as shown:

```
from tkinter import *

class MyAnchorJustify(Tk):
    def __init__(self):
        super().__init__()
        self.title('Anchor and Jusify in label')
        myl1 = Label(self, text = 'Anchor and Justify in Right')
        myl1.pack()
```

```

myl2= Label(self,
            text = 'Stay\nSafe Safe\
nSafe Safe Safe',bd = 2, relief = 'solid',font = ('Times New Ro-
man',12),
            width = 20,height = 4,anchor = NE, justi-
fy = RIGHT)
myl2.pack()
myl3 = Label(self, text = 'Anchor and Justify in Left')
myl3.pack()
myl4= Label(self,text = 'Stay\nSafe Safe\
nSafe Safe Safe', bd = 2, relief = 'solid',
            font = ('Times New Ro-
man',12),width = 20,height = 4,anchor = NE,justify = LEFT)
myl4.pack()

if __name__ == "__main__":
    myroot = MyAnchorJustify()
    myroot.geometry('350x300')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.6*:

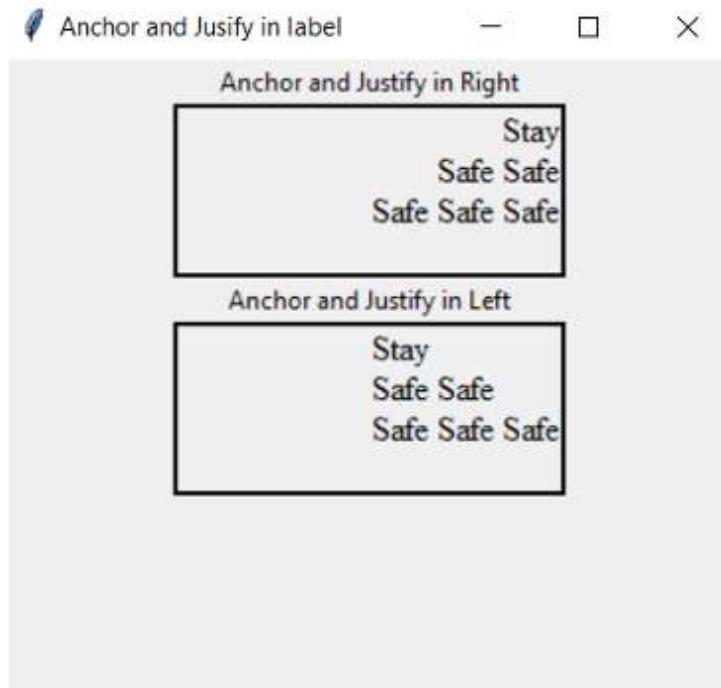


Figure 5.6: Output

Note: The preceding code is covered in Program Name: Chap5_Example6.py

Moreover, we can access the options of a **tkinter** label:

```

from tkinter import *

class MyAcessOption(Tk):
    def __init__(self):
        super().__init__()
        self.title('Access options of a tkinter label')
        myl2= Label(self,text = 'Stay\nSafe From\nFrom Corona Vi-
russ', bd = 2, bg = 'LightGreen', relief = 'sol-
id',
                    font = ('Arial',14),width = 20,height = 4,an-
chor = NW,justify = LEFT)
        myl2.pack()
        print(myl2["text"])
        print("-----")
        print(myl2["bd"])
        print(myl2["bg"])
        print(myl2["font"])
        print(myl2["width"])
        print(myl2["height"])
        print(myl2["anchor"])
        print(myl2["justify"])

if __name__ == "__main__":
    myroot = MyAcessOption()
    myroot.geometry('350x150')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.7*:

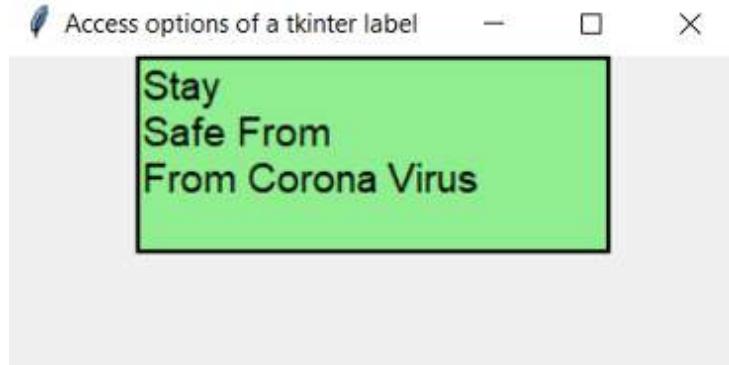


Figure 5.7: Output of

Here is the output at the console:

```
Stay
Safe From
From Corona Virus
-----
2
LightGreen
Arial 14
20
4
nw
left
```

Note: The preceding code is covered in Program Name: Chap5_Example7.py

So, we can access the options associated with a label by using key-value pair.

Now, just like we can access the options of a label, there is a provision for dynamically changing label options, as shown:

```

from tkinter import *

class MyAccessChangeOption(Tk):
    def __init__(self):
        super().__init__()
        self.title('Access and Change options of a tkinter label')
        myl2= Label(self,text = 'Stay\nSafe From\nFrom Corona Virus', bd = 2, bg = 'LightGreen', fg = 'Yellow',
                   relief = 'solid',
                   font = ('Arial',14),width = 20,height = 4,anchor = NW,justify = LEFT)
        myl2.pack()
        myl2['bg'] = 'LightBlue' # we are changing bg to LightBlue
        myl2['fg'] = 'Red'# we are changing fg to Red

if __name__ == "__main__":
    myroot = MyAccessChangeOption()
    myroot.geometry('450x130')
    myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.8*:

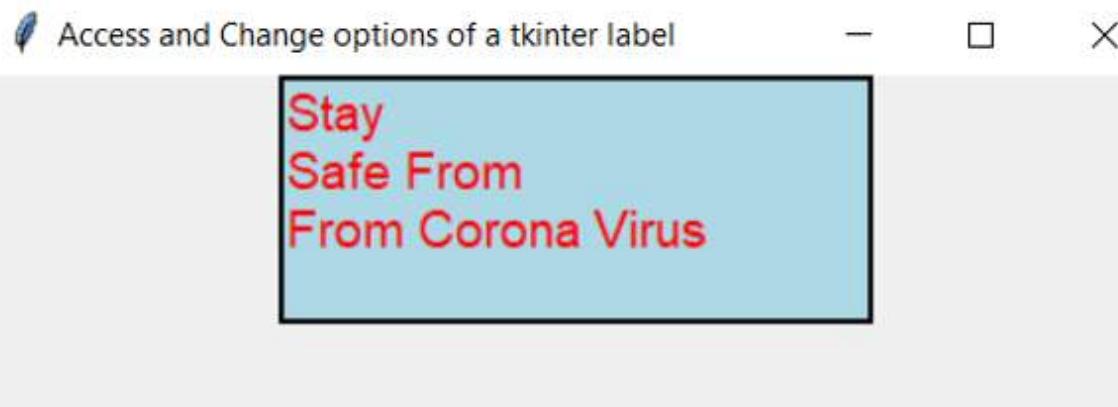


Figure 5.8: Output

Note: The preceding code is covered in Program Name: Chap5_Example8.py

We can also display the default values (key values) of a label whether it is mentioned or not, as shown:

```
from tkinter import *

class MyAccessChangeOption(Tk):
    def __init__(self):
        super().__init__()
        self.title('Displaying key values of a tkinter label')
        myl2= Label(self,text = 'Python', bd = 2, bg = 'Light-Blue',relief = 'solid',font = ('Verdana',12),
                    width = 12,height = 4,anchor = SE,justify = RIGHT)
        myl2.pack()

    for loop in myl2.keys():
        print(loop,':',myl2[loop]) # will display default values if not mentioned.

if __name__ == "__main__":
    myroot = MyAccessChangeOption()
    myroot.geometry('450x130')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.9*:

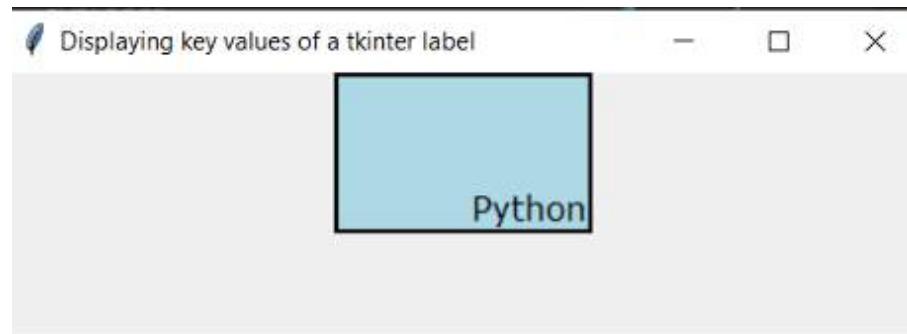


Figure 5.9: Output of Chap5_Example9.py in GUI

Here is the output at the console:

```
activebackground : SystemButtonFace
activeforeground : SystemButtonText
anchor : se
background : LightBlue
bd : 2
bg : LightBlue
bitmap :
borderwidth : 2
compound : none
cursor :
disabledforeground : SystemDisabledText
fg : SystemButtonText
font : Verdana 12
foreground : SystemButtonText
height : 4
highlightbackground : SystemButtonFace
highlightcolor : SystemWindowFrame
highlightthickness : 0
image :
justify : right
padx : 1
pady : 1
relief : solid
state : normal
takefocus : 0
```

```
text : Python
textvariable :
underline : -1
width : 12
wraplength : 0
```

Note: The preceding code is covered in Program Name: Chap5_Example9.py

So, we can see that all the key method values will be displayed in the console. If not mentioned, then default values will be shown.

We can use **textvariable** option as it will associate a **tkinter** variable to the label widget, that is, we can use **StringVar()** and **textvariable** for a tkinter label widget, as shown:

```
from tkinter import *

class MyStringVartext(Tk):
    def __init__(self):
        super().__init__()
        self.myval1 = StringVar()
        self.title('StringVar() and textvariable a tkinter label')
        self.myl2= Label(self,font = 'Helvetica',textvariable= self.
myval1,relief = 'groove')
        self.myl2.pack()
        self.myval1.set('python is awesome')

if __name__ == "__main__":
    myroot = MyStringVartext()
    myroot.geometry('400x130')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.10*:



Figure 5.10: Output

Note: The preceding code is covered in Program Name: **Chap5_Example10.py**

We have already discussed how to use **StringVar()** and **textvariable**. This example is just a recap of how to use it.

We can set the text of a label using either the key-value pair approach or by using the **set()** method, as shown:

```
from tkinter import *

class MyStringVar_key_text(Tk):
    def __init__(self):
        super().__init__()
        self.myval1 = StringVar()
        self.title('StringVar() and textvariable a tkinter label')
        self.myl2= Label(self,font = 'Helvetica',textvariable= self.
myval1,relief = 'groove')
        self.myl2.pack()
        self.myl3= Label(self,font = ('Arial',12),text= 'Hello',re-
lief = 'groove')
        self.myl3.pack(padx = 10, pady = 10)
        self.myl3['text'] = 'Key/value pair approach of set-
ting text'
        self.myval1.set('using textvariable set')

if __name__ == "__main__":
    myroot = MyStringVar_key_text()
    myroot.geometry('400x100')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.11*:

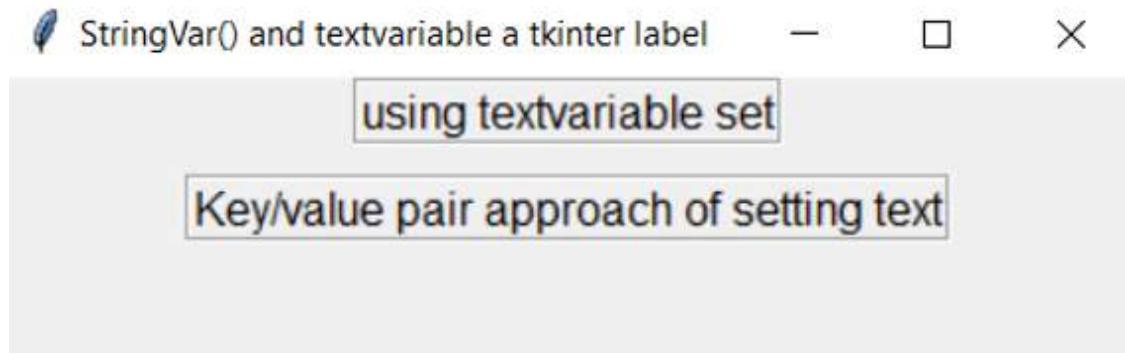


Figure 5.11: Output

Note: The preceding code is covered in Program Name: Chap5_Example11.py

In the above example, we can see how the text of a label is changed using **StringVar()** and the key-value pair approach.

We can display images using the label, as shown:

```
from tkinter import *

class MyImage(Frame):
    def __init__(self, root = None):
        Frame.__init__(self, root)
        self.root = root
        self.myphoto = PhotoImage(file = 'butterfly1.gif')
        self.myl1 = Label(self.root,image = self.myphoto)
        self.myl1.pack(padx = 10, pady = 10)

if __name__ == "__main__":
    myroot = Tk()
    myobj = MyImage(myroot)
    myroot.title('Image using label')
    myroot.geometry('300x300')
    myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.12*:

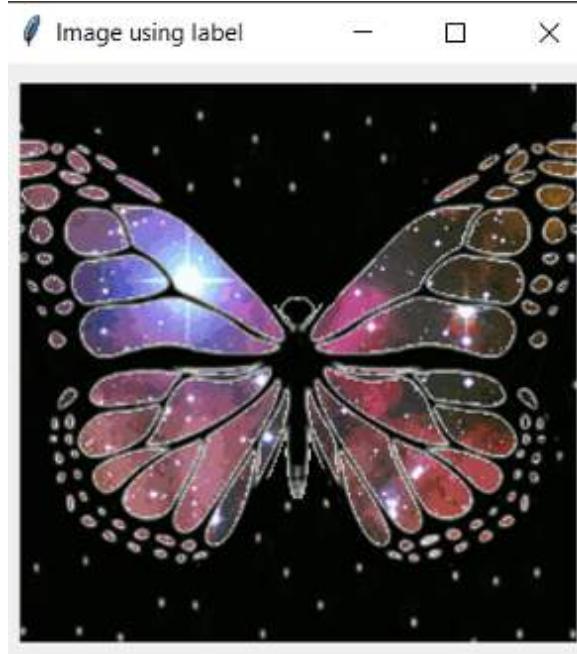


Figure 5.12: Output

Note: The preceding code is covered in Program Name: Chap5_Example12.py

tkinter Message Widget

This widget will display the text messages to the user which cannot be edited. It contains more than one line and can only be shown in a single font.

The syntax is as follows:

```
mymsg1= Message(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas, are bg, bd, bitmap, cursor, anchor, fg, font, width, height, image, justify, padx, pady, relief, text, textvariable, underline, wraplength, and width.

We have seen and learned all the options till now. Let us see the example directly:

```
from tkinter import *

myroot = Tk()
mystr = StringVar()

# creation of message widget object
mymsg1 = Message( myroot, textvariable=mystr, relief=RAISED, font = ('Calibri',12), fg = 'Red', bg = 'LightGreen' )

mystr.set("This is a string message")
mymsg1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.13*:

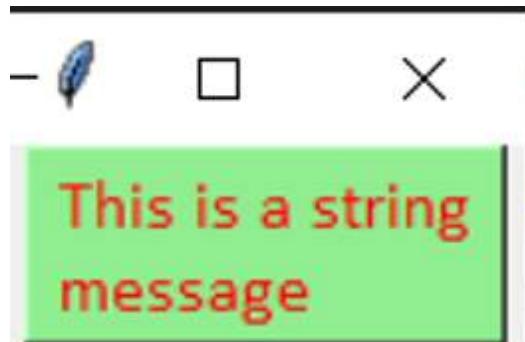


Figure 5.13: Output

Note: The preceding code is covered in Program Name: Chap5_Example13.py

We can also use the text option to display the message, as shown:

```

from tkinter import *

myroot = Tk()

mytxt = 'Stay Safe from Corona Virus. Follow social distancing Please.:)'

# creation of message widget object
mymsg1 = Message( myroot, text=mytxt, relief=RAISED, font = ('Calibri',12), fg = 'Red', bg = 'LightGreen')

mymsg1.pack()
myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.14*:

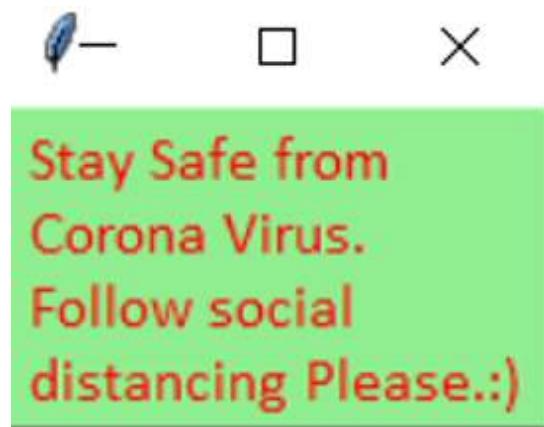


Figure 5.14: Output

Note: The preceding code is covered in Program Name: Chap5_Example14.py

We shall now see how to use Statusbar in Python tkinter GUI applications using label widget.

The label widget will display a narrow bar at the GUI bottom for indicating some extra information like file word counts or some relevant information which will add some extra value when interfacing with the user. There is no

dedicated status bar widget in tkinter but a label widget with an appropriate configuration, that can be worked like a status bar in the GUI applications. Refer to the following:

```
from tkinter import *

myroot = Tk()
myroot.geometry('350x200')
myroot.title("StatusBarExample")

mystatusbar = Label(myroot, text="It is a statusbar example...", bd=1, relief=SUNKEN, anchor=W) # where bd: bordersize , relief: label appearance, anchor: text alignment within the label

mystatusbar.pack(side=BOTTOM, fill=X) # positioned at the GUI bottom and covers the whole window width if window is resized
myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.15*:

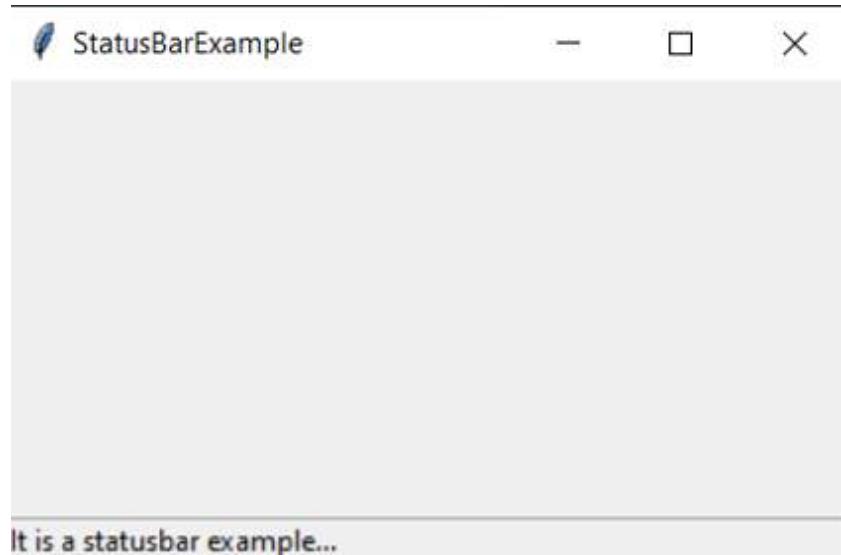


Figure 5.15: Output

Note: The preceding code is covered in Program Name: Chap5_Example15.py

tkinter MessageBox Widget

This widget will display the message boxes in the Python applications. The relevant messages are displayed with various functions depending on the application requirements.

The syntax is as follows:

```
messagebox.function_name(title, message [, options])
```

where,

- **function_name:** It is the appropriate message box function name.
- **title:** This parameter can be used to display custom string in the title box.
- **message:** This parameter can be used to display custom string as message on the message box.
- **options:** The options used are default and parent. The default option will mention the default button types like ABORT, RETRY or IGNORE in the messagebox. The parent option will specify the window on top of which displays the messagebox.

Now, we shall see different functions for displaying the appropriate message boxes.

showinfo()

This **messagebox**, when used, will display the relevant information to the user, as shown:

```

from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('Showinfo example')

def mydisplay():
    messagebox.showinfo("Showinfoexample","This is a basic showinfo example")
    mybtn1 = Button(myroot, text = 'ClickShowInfo', command = mydisplay)
    mybtn1.pack()
myroot.mainloop()

```

Output:

The output can be seen in *Figure 5.16*:

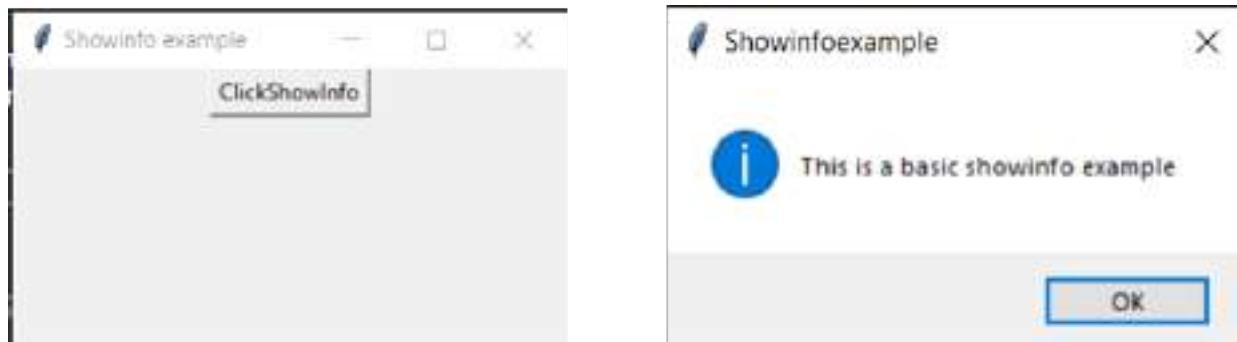


Figure 5.16: Output

Note: The preceding code is covered in Program Name: Chap5_Example16.py

In this code, **Showinfoexample** is a title in the box and “**This is a basic showinfo example**” is the displayed information when the **ClickShowInfo** button is clicked. So, using the above function depicts plain information.

showwarning()

This function will display a warning message to the user, as shown:

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('Warningmessage')

def mydisplay():
    messagebox.showwarning("ShowWarningexample","This is a ba-
    sic warning message example")

mybtn1 = Button(myroot, text = 'ClickWarningMsg', com-
    mand = mydisplay)
mybtn1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.17*:

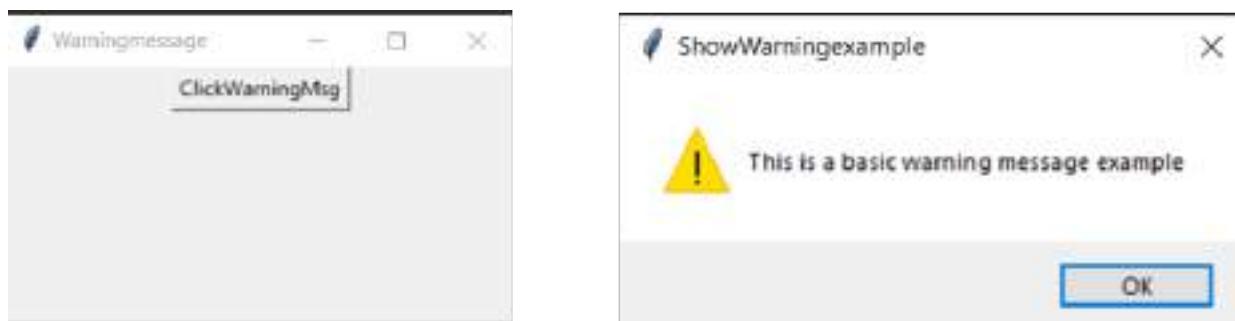


Figure 5.17: Output

Note: The preceding code is covered in Program Name: Chap5_Example17.py

In this code, we have displayed a message box using a `showwarning()` function. The **ShowWarningexample** is a title and “**This is a basic warning message example**” is the warning information when the **ClickWarningMsg** button is clicked.

showerror()

This function, when used, will display the error message to the user, as shown:

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('Errormessage')

def mydisplay():
    messagebox.showerror("Showerrorexample", "This is a basic error message example")

mybtn1 = Button(myroot, text = 'ClickErrorMsg', command = mydisplay)
mybtn1.pack()
myroot.mainloop()
```

Output:

The output can be seen in *Figure 5.18*:

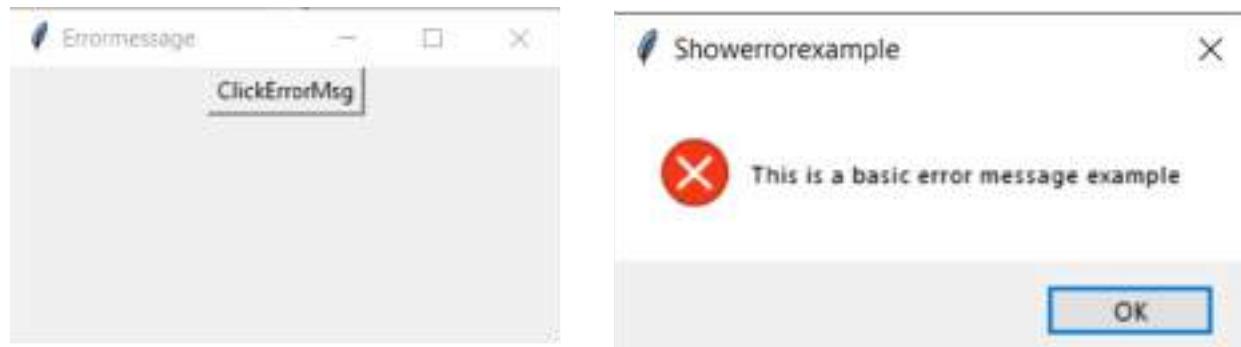


Figure 5.18: Output

Note: The preceding code is covered in Program Name: Chap5_Example18.py

In this code, we have displayed a message box using a `showerror()` function. The **Showerrorexample** is a title and “**This is a basic error message**

example" is the error information when the **ClickErrorMsg** button is clicked.

askquestion()

This function can be used to display custom confirmatory questions framed by user. The questions can be used for validation or getting user confirmation. The accepted answer will be either yes or no.

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('AskQuestion')

def mydisplay():
    ans = messagebox.askquestion("AskQuestion example","Do you want to continue")
    if ans == 'yes':
        messagebox.showinfo('Message','You have chosen Yes')
    else:
        messagebox.showinfo('Message','You have chosen No')

mybtn1 = Button(myroot, text = 'ClickAskMsg', command = mydisplay)
mybtn1.pack()
myroot.mainloop()
```

Output when the ClickAskMsg button is clicked:

The output can be seen in *Figure 5.19*:

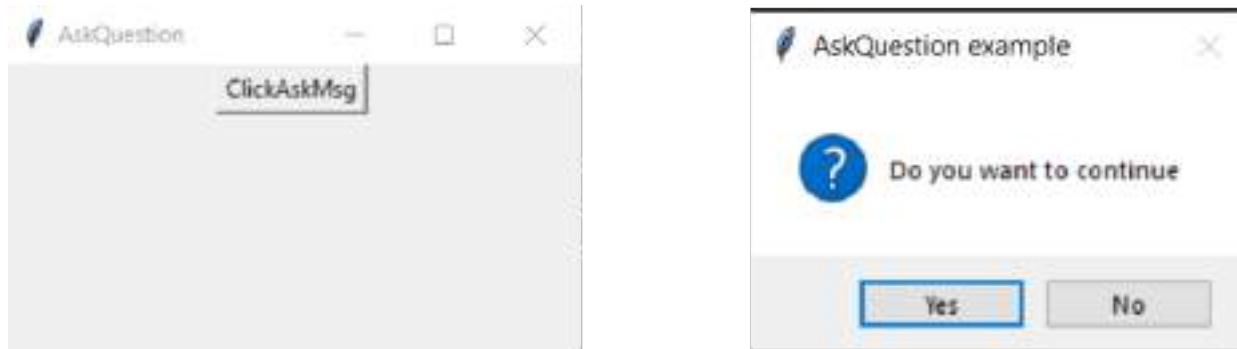


Figure 5.19: Output when ClickAskMsg button is clicked

Output when the Yes button is clicked:

The output can be seen in [Figure 5.20](#):

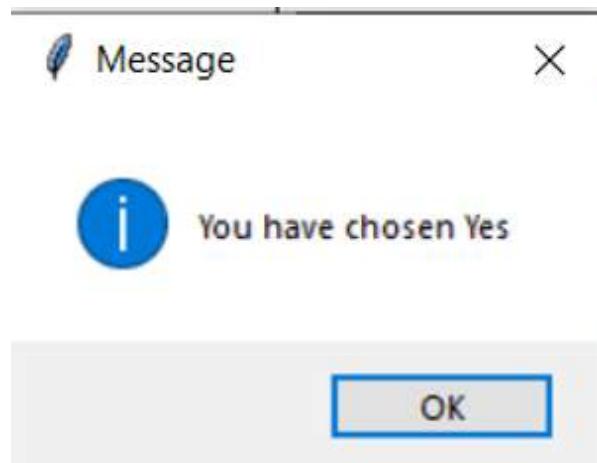


Figure 5.20: Output when Yes button is clicked

Output when the No button is clicked:

The output can be seen in [Figure 5.21](#):

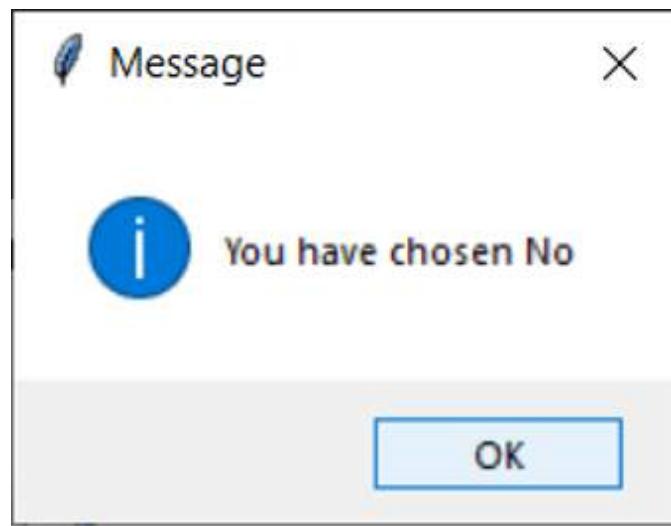


Figure 5.21: Output when No button is clicked

Note: The preceding code is covered in Program Name: Chap5_Example19.py

askokcancel()

This function, when used, will confirm the user's responses regarding the application activity. Here, the answers are OK and Cancel. Refer to the following:

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('AskOkCancel')
```

```
def mydisplay():
    messagebox.askokcancel("AskOkCancel example", "Redirect-
ing to www.abc.com")

mybtn1 = Button(myroot, text = 'ClickOkCancelMsg', com-
mand = mydisplay)
mybtn1.pack()
myroot.mainloop()
```

Output when the ClickOkCancelMsg button is clicked:

The output can be seen in *Figure 5.22*:

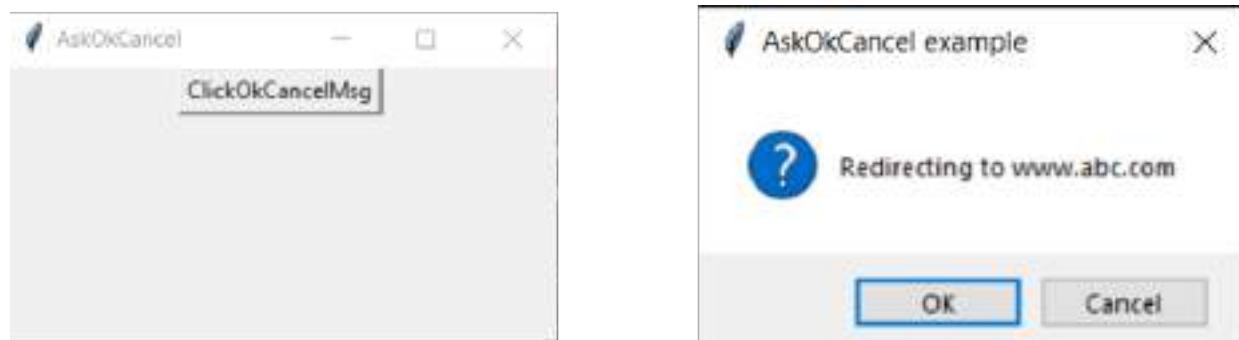


Figure 5.22: Output of Chap5_Example20.py

Note: The preceding code is covered in Program Name: Chap5_Example20.py

askyesno()

This function, when used, will ask the user some questions which can be answered by using yes or no. Here, the answers are Yes and No. Refer to the following:

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('AskYesNo')

def mydisplay():
    messagebox.askyesno("AskYesNo example", "Will you do it")

mybtn1 = Button(myroot, text = 'ClickYesNoMsg', command = mydisplay)

mybtn1.pack()
myroot.mainloop()
```

Output when the ClickYesNoMsg button is clicked:

The output can be seen in *Figure 5.23*:

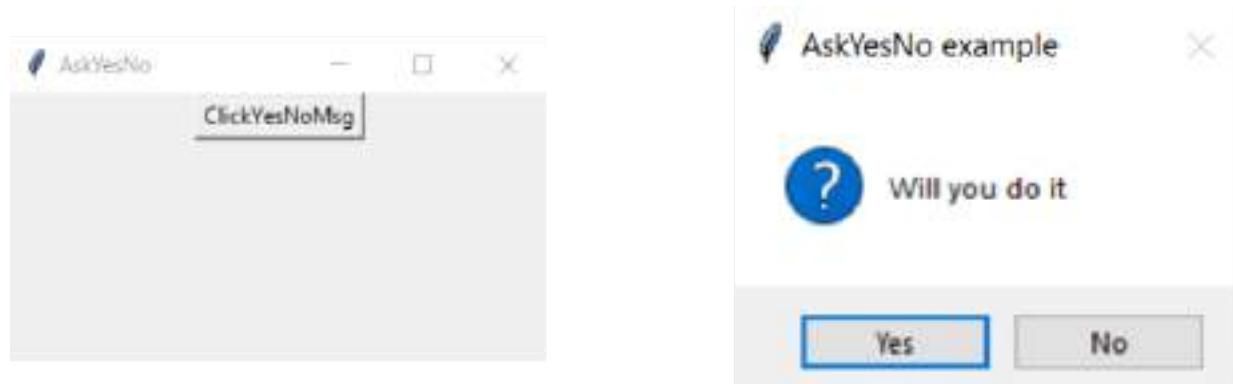


Figure 5.23: Output

Note: The preceding code is covered in Program Name: Chap5_Example21.py

askretrycancel()

This function, when used, will ask the user to perform a particular task again or not. Here, the answers are Retry and Cancel. Refer to the following:

```
from tkinter import *
from tkinter import messagebox

myroot = Tk()
myroot.geometry("300x150")
myroot.title('AskRetryCancel')

def mydisplay():
    messagebox.askretrycancel("AskRetryCancel example","Will you do it")

mybtn1 = Button(myroot, text = 'ClickRetryCancelMsg', command = mydisplay)
mybtn1.pack()
myroot.mainloop()
```

Output when the ClickRetryCancelMsg button is clicked:

The output can be seen in *Figure 5.24*:

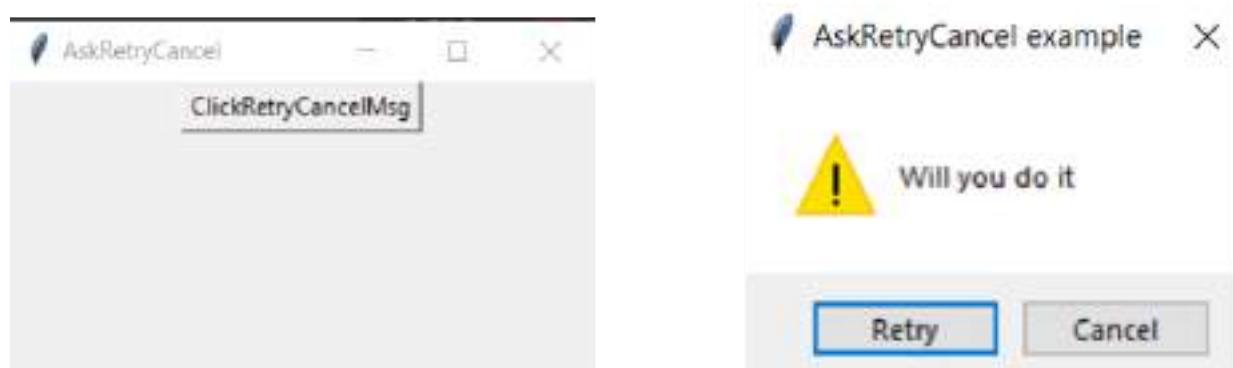


Figure 5.24: Output

Note: The preceding code is covered in Program Name: Chap5_Example22.py

Conclusion

In this chapter, we learned about the creation of a simple GUI app using the tkinter Label widget, which depicts the ways of displaying a text or image on a window form. Using tkinter Label widget, we saw how to position its text, how to pad space around the text, how to justify the text, how to both anchor and justify the text simultaneously, how to access the widget options using a key-value pair, how to dynamically change label options, how to display the default values, and how to display image in the widget using examples.

Moreover, we explored how to use this label widget as a statusbar with code. We have used text option to display the message for demonstrating tkinter Message widget. Furthermore, we viewed a display of prompt unedited text messages to the user with this tkinter Message widget. Finally, we looked into multiple message boxes such as information, warning, error, question, okcancel, yesno and so on, in a Python application by using the tkinter MessageBox widget.

Points of remember

- Using a label widget, we can show text or an image in a window. The Label widget's text, font, foreground, and border are some of its crucial properties.
- Text can be automatically wrapped when displayed in a window using a message widget. The Message widget's text, font, foreground, background, and border are some of its crucial properties.
- A message box with optional buttons and a message can be displayed with the MessageBox Widget. The messagebox is most frequently used to provide an alert or confirmation message to the user. The message box's type argument can be used to specify the buttons that are available.
- Other widgets, such as photos, buttons, and other labels, can be contained inside the Label widget.
- The MessageBox widget offers a number of message box types, each with a unique function and return result, including showinfo, showwarning, showerror, askquestion, askokcancel, and askyesno.

- The button that the user pressed is indicated by the value the MessageBox widget delivers. To determine the user's choice, the value can be compared to predefined constants as *yes*, *no*, *ok*, *cancel*, *yesno*, and so on.

Questions

1. Explain the tkinter Label widget in detail.
2. Explain the tkinter widget that is used for displaying text or an image on the screen.
3. Write a program to display “Python is Awesome” on the screen.
4. Explain the usage of the Message widget in detail.
5. Which widget is used to display the text messages to the user, which cannot be edited? Explain with an example.
6. Write a program to display “This is a text” using the Message widget.
7. Explain tkinter MessageBox Widget and its syntax.
8. Explain any three functions and their use of the tkinter MessageBox Widget.
9. Write short notes on the following:
 - a. showerror()
 - b. askquestion()
 - c. askokcancel()
 - d. askyesno()

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 6

Getting Insights of Container Widgets in tkinter

Introduction

The need for container widgets in Python tkinter arises from their ability to manage and organise the layout of other widgets in a **Graphical User Interface (GUI)** application. We can group relevant widgets together and arrange them in a way that makes sense to the user by utilising container widgets. The following are some advantages of using container widgets in tkinter:

- We can manage the positioning and layout of child widgets using container widgets like tkinter Frame and tkinter PanedWindow, which makes it simpler to design a logical and visually appealing GUI.
- The container widgets are used to organize your code, which will make it simpler to read and maintain. The code can be made simpler by grouping related widgets and defining them as a single entity.
- The container widgets help us in logically grouping widgets depending on their intended use. For instance, we could use a tkinter LabelFrame to group a set of radio buttons related to one particular option.
- The GUI can be scaled and resized with the use of container widgets. The tkinter **PanedWindow** widget can be used to divide the GUI into resizable panes. The tkinter Notebook widget is used to provide a tabbed interface.

Let us view different container widgets in tkinter.

Structure

In this chapter, we will discuss the following topics:

- tkinter Frame Widget
- tkinter LabelFrame Widget
- tkinter Tabbed/Notebook Widget
- tkinter PanedWindow Widget
- tkinter Toplevel Widget

Objectives

After going through this chapter, the user will learn about tkinter Frame widget where different widget positions can be arranged, padding can be provided, can be used as a geometry manager for other widgets and so on. We shall look into the variant of the Frame widget, which is tkinter LabelFrame and is a container for complex window layouts. Users will be able to see frame features along with label display. Moreover, we shall view creating a tabbed widget with the help of the Notebook widget. Here, the user can select different pages of content by clicking on tabs. The importance of the tkinter PanedWindow widget will be explored where multiple examples will be seen containing horizontal or vertical stacks of child widgets. Finally, we will look into the tkinter Toplevel widget where the concepts are being explained for the creation and display of top-level windows.

tkinter Frame Widget

This widget is a container widget responsible for arranging widgets positions. It is used as a geometry master for other widgets, but only one per frame and is a rectangular region of the screen. The other widgets are grouped into complex layouts using the **Frame** widget. If we need to provide padding between the widgets, then it can be done by using this widget. Whenever we will be implementing compound widgets, then a **Frame** class can act as a base class. We can have multiple frames per window.

The syntax is as follows:

```
myfr1= Frame(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas, are bg, bd, height, cursor, highlightthickness, highlightcolor, highlightbackground, width, and relief.

We have discussed all the options as we are aware. Let us now directly see the examples:

```
from tkinter import*

myroot=Tk()
myroot.geometry("300x300")

myframe1=Frame(myroot, width=150, height=150, bg="Red")
myframe1.grid(row=0, column=0)

myframe2=Frame(myroot, width=150, height=150, bg="Green")
myframe2.grid(row=1, column=0)

myframe3=Frame(myroot, width=150, height=150, bg="Blue")
myframe3.grid(row=0, column=1)

myframe4=Frame(myroot, width=150, height=150, bg="Cyan")
myframe4.grid(row=1, column=1)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.1*:

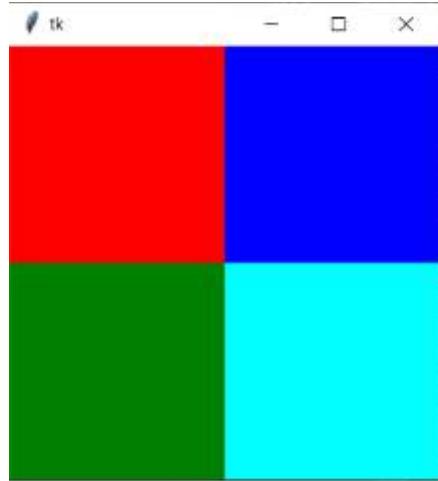


Figure 6.1: Output

Note: The preceding code is covered in Program Name: Chap6_Example1.py

In this code, we have created multiple frames in a window which are 4 here. The **grid()** method will align the tkinter frames in rows and columns. Different Frame objects are created viz myframe1, myframe2, myframe3 and myframe4. The width and height of each frame are being set and background color option will provide the necessary background color to the frame.

Now, we shall view to arrange label, **Entry**, and **button** widgets in a single frame on a parent widget:

```
from tkinter import*

myroot=Tk()
myroot.geometry("220x100")

myframe1=Frame(myroot)
myl1 = Label(myframe1, text = 'Name')
myl1.grid(row = 0, column = 0)
myl2 = Label(myframe1, text = 'Age')
myl2.grid(row = 1, column = 0)
myl3 = Label(myframe1, text = 'PhoneNumber')
myl3.grid(row = 2, column = 0)

mye1 = Entry(myframe1)
mye1.grid(row = 0, column = 1)
mye2 = Entry(myframe1)
mye2.grid(row = 1, column = 1)
mye3 = Entry(myframe1)
mye3.grid(row = 2, column = 1)

mybtn = Button(myframe1, text = 'View')
mybtn.grid(row = 3, columnspan = 2)
myframe1.grid(row=0, column=0)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.2*:

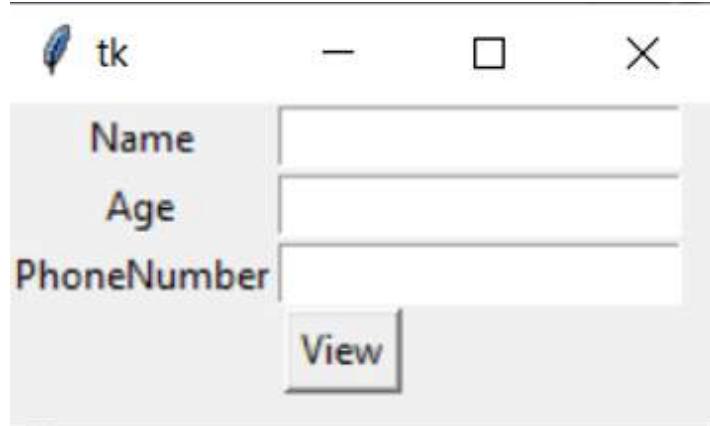


Figure 6.2: Output

Note: The preceding code is covered in Program Name: Chap6_Example2.py

In this code, there is only one frame and different widgets, namely label, Entry, and button. The locations of these widgets in a frame have been positioned.

We can also create another frame and can place it on a parent widget as shown:

```
from tkinter import*

myroot=Tk()
myroot.geometry("430x100")

# myframe1 with row = 0, column = 0
myframe1=Frame(myroot)
myframe1.grid(row=0, column=0)

myl1 = Label(myframe1, text = 'Name')
myl1.grid(row = 0, column = 0)
myl2 = Label(myframe1, text = 'Age')
myl2.grid(row = 1, column = 0)
myl3 = Label(myframe1, text = 'PhoneNumber')
myl3.grid(row = 2, column = 0)

mye1 = Entry(myframe1)
mye1.grid(row = 0, column = 1)
mye2 = Entry(myframe1)
mye2.grid(row = 1, column = 1)
mye3 = Entry(myframe1)
mye3.grid(row = 2, column = 1)
```

```
mybtn = Button(myframe1, text = 'View')
mybtn.grid(row = 3, columnspan = 2)

# mysideframe1 with row = 0, column = 1
mysideframe1=Frame(myroot)
mysideframe1.grid(row=0, column=1, padx = 20)

myl1 = Label(mysideframe1, text = 'Sex')
myl1.grid(row = 0, column = 0)
myl2 = Label(mysideframe1, text = 'City')
myl2.grid(row = 1, column = 0)
myl3 = Label(mysideframe1, text = 'Address')
myl3.grid(row = 2, column = 0)

mye1 = Entry(mysideframe1)
mye1.grid(row = 0, column = 1)
mye2 = Entry(mysideframe1)
mye2.grid(row = 1, column = 1)
mye3 = Entry(mysideframe1)
mye3.grid(row = 2, column = 1)

mybtn = Button(mysideframe1, text = 'Display')
mybtn.grid(row = 3, columnspan = 2)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.3*:



Figure 6.3: Output

Note: The preceding code is covered in Program Name: Chap6_Example3.py

In this code, we have created 2 frames, one in row=0, column = 0, and another in row = 0, column = 1. All the different widgets are placed inside these frames as per need.

tkinter LabelFrame Widget

This widget will act like a container for grouping the number of interrelated widgets and will draw a border around its child widgets. The title can be displayed for the above widgets. It is a variant of the **Frame** widget having all the frame features.

The syntax is as follows:

```
mylf1= LabelFrame(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, font, cursor, height, highlightbackground, highlightcolor, highlightthickness, labelAnchor, padx, pady, relief, text, width, container, labelwidget, and colormap.

We have seen most of the options but some undiscussed options are as follows:

- **labelanchor**: This option will represent the exact text position within the widget. The default is NW.
- **container**: This option will make the **LabelFrame** become a container widget when set to True. The default value is False.
- **labelwidget**: This option allows the user to choose the widget used for the label. The text is used for the label by the frame when no value is specified.
- **colormap**: This option will specify colormap (which means 256 colors will be used to form the graphics) to be used for the above widget.

We shall see some examples for better understanding:

```
from tkinter import *

class MyLabelFrame(Tk):
    def __init__(self):
        super().__init__()

        # labelframe is defined and the text is assigned to be displayed by the frame.
        self.mylf1 = LabelFrame(self, text="Welcome to La-
```

```

bel and ButtonFrame ", font = ('Calibri',12), bg = 'LightBlue')
    self.mylf1.pack(fill="both", expand="yes")

        #Label is defined and created
        self.myl1 = Label(self.
mylf1, text="I am Label", bg = 'Magenta')
        self.myl1.pack(side = TOP)

        #Button is defined and created
        self.mybtn1 = Button(self.
mylf1, text="I am Button", bg = 'Violet')
        self.mybtn1.pack(side = LEFT)

        # labelframe is defined and the text is assigned to be displayed by the frame.
        self.mylf2 = LabelFrame(self, text="Welcome to CheckButton and Radiobutton Frame", font = ('Calibri',12), bg = 'LightGreen')
        self.mylf2.pack(fill="both", expand="yes")

        #Checkbutton is defined and created
        self.mychk1 = Checkbutton(self.
mylf2, text="I am CheckButton", bg = 'Pink')
        self.mychk1.pack(side = RIGHT)

        #RadioButton is defined and created
        self.myr1 = Radiobutton(self.
mylf2, text="I am RadioButton", bg = 'Brown')
        self.myr1.pack(side = BOTTOM)

if __name__ == '__main__':
    myroot = MyLabelFrame() # creating an instance of MyLabelFrame
    myroot.geometry('400x150')
    myroot.mainloop() # infinite loop to run the application

```

Output:

The output can be seen in *Figure 6.4*:

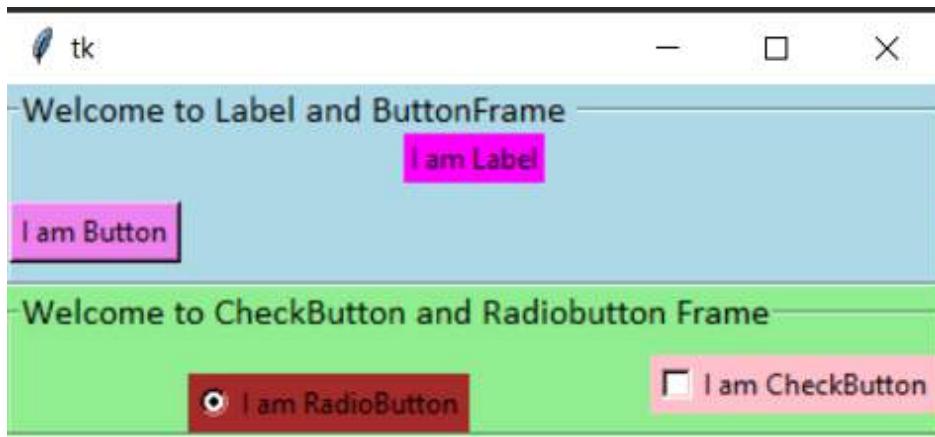


Figure 6.4: Output

Note: The preceding code is covered in Program Name: Chap6_Example4.py

We can put the text used for the label in any anchor position as shown:

```

from tkinter import *

class MyLabelFrame(Tk):
    def __init__(self):
        super().__init__()

        # labelframe is defined and the text is assigned to be displayed by the frame.
        self.myf1 = LabelFrame(self, text="I am LabelFrame", font = ('Calibri',12),
                                bg = 'LightBlue', labelanchor = E)
        self.myf1.pack(fill="both", expand="yes")

        #Label is defined and created
        self.myl1 = Label(self.myf1, text="I am Label", bg = 'Magenta')
        self.myl1.pack(side = LEFT)

if __name__ == '__main__':
    myroot = MyLabelFrame() # creating an instance of Scrollbar_Entry
    myroot.geometry('400x150')
    myroot.mainloop() # infinite loop to run the application

```

Output:

The output can be seen in *Figure 6.5*:

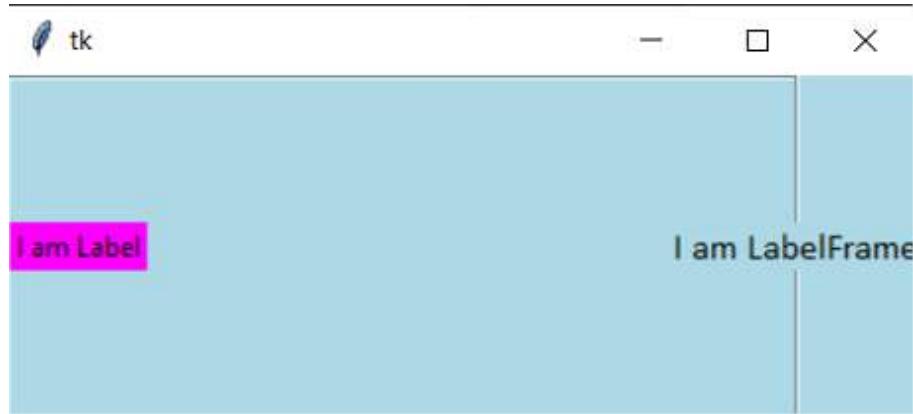


Figure 6.5: Output

Note: The preceding code is covered in Program Name: Chap6_Example5.py

In the preceding code, the text on the label is anchored to the East.

So, we can say that **LabelFrame** is a combination of Label and Frames with more Label attributes in it.

tkinter Tabbed/Notebook Widget

The tabbed widget is created by using the Notebook widget of the **ttk** module. Modern-looking **Graphical User Interfaces (GUIs)** can be made using the improved range of widgets and styles offered by the **ttk** module in Tkinter. The term "Themed Tkinter" refers to a style of tkinter widgets that is more unified and aesthetically pleasing than the standard Tkinter widgets. As the underlying tkinter framework is built on top of the **ttk** module, user can still utilize popular tkinter methods and properties with **ttk** widgets. By doing this, the user may take advantage of the extra functionality and aesthetic choices offered by **ttk** while still keeping the compatibility of the previous tkinter code.

Compared to the regular tkinter widgets, the **ttk** widgets offer a variety of advantages, including:

- Rendering of anti-aliased fonts in X11.
- Transparency of windows (X11 only; requires composite window manager).
- Separation between the code responsible for a widget's look and that responsible for its behavior.

This **ttk** notebook widget will manage the windows collection, and display one at a time. The child window will be associated with a tab. A single tab can be selected by the user at a time to view the window content.

The syntax is as follows:

mytabcontrol= Notebook(myroot, options...)

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are padding, compound, sticky, underline, and text.

We have seen most of the options but some undiscussed options are as follows:

- **compound**: This option will supply both image and text to be displayed on the tab. It can take one of the three values: none, left (image displayed to the left of the text) or right (image displayed to the right of the text).
- **padding**: This option will add extra space around all 4 sides of the panel's content.

We shall see some examples:

```
from tkinter import *
from tkinter import ttk

myroot = Tk()
myroot.title("Demo Tab Widget")
mytabcontrol = ttk.Notebook(myroot) # L1

mytab1 = ttk.Frame(mytabcontrol) # L2
mytab2 = ttk.Frame(mytabcontrol)

mytabcontrol.add(mytab1, text ='MyTab1') # L3
mytabcontrol.add(mytab2, text ='MyTab2')

mytabcontrol.pack(expand = 1, fill ="both") # L4

ttk.Label(mytab1, text ="Welcome to Tab1", font = ('Helvetica',12)).grid(column = 0, row = 0, padx = 50, pady = 50) # L5
ttk.Label(mytab2, text ="I hope u now understood the tab concept now", font = ('Times New Roman',12)).grid(column = 0, row = 0, padx = 50, pady = 50)

myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.6*:

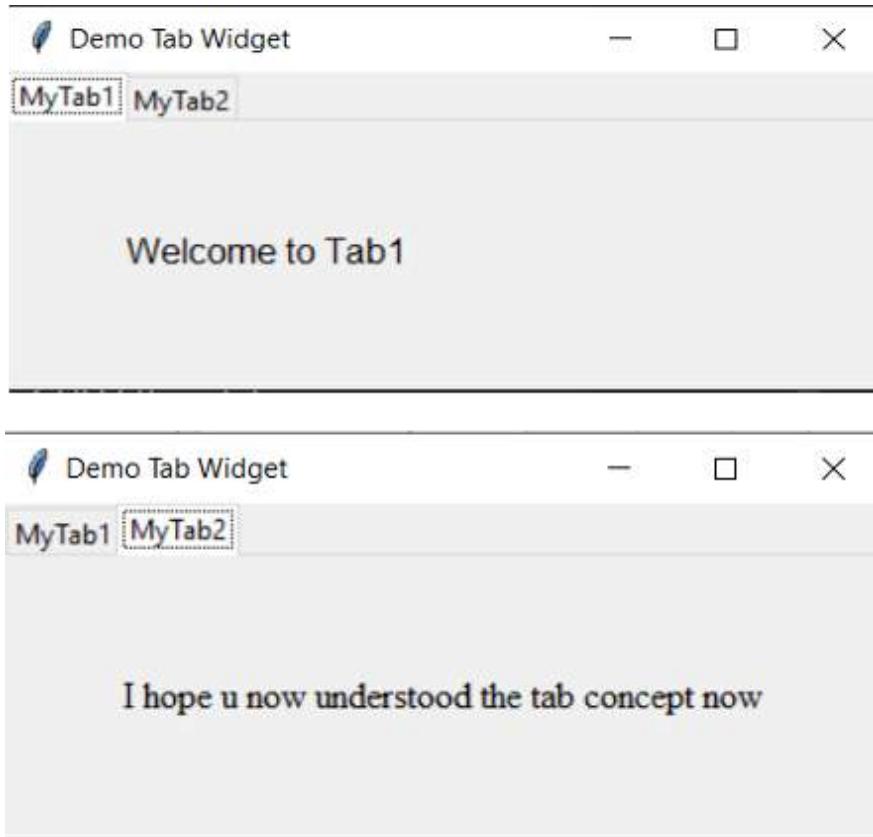


Figure 6.6: Output

Note: The preceding code is covered in Program Name: Chap6_Example6.py

In this code, we have initially imported the tkinter **ttk** module, containing the **Notebook** widget, followed by the creation of a parent window. We have given title to the parent window.

In L1, we have created a tab control.

In L2, we have created the tabs by using Frames which act like a container and will be grouping the tab widgets.

In L3, we have added the tabs where mytab1 and mytab2 are the child widgets of **tabcontrol** and the above method is present in **tk.ttk.Notebook** class. So, it will add new tabs to the **Notebook** widget.

In L4, the widgets will be organized in blocks before placing them in the parent widget and different options such as fill and expand are used.

In L5, we have created label widgets that will display text on the screen, and its position is specified on the parent window.

tkinter PanedWindow widget

This widget is a container widget that can have any number of child widgets (panes) and can be arranged either vertically or horizontally. Each child pane can be resized by moving the separator lines called **sashes** by using the mouse. Whenever there is a requirement to implement different layouts in the Python applications, we can go for this widget.

The syntax is as follows:

```
mypw1= PanedWindow(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, borderwidth, cursor, handlepad, height, handlesize, relief, orient, sashcursor, sashrelief, width, sashwidth, and showhandle.

We have seen most of the options but some undiscussed options are as follows:

- **handlepad**: This option will represent the distance between the handle and sash end having the default size of 8 pixels. If the orientation is horizontal, it is the distance between the handle and sash top.
- **handlesize**: This option will represent the handle size having the default size of 8 pixels.
- **orient**: This option will allow placing the child Windows at different positions in a frame. If set to horizontal, the child windows will be placed side by side. If set to vertical, then child windows will be placed from top to bottom.
- **sashpad**: This option will allow padding to be done around each sash.

- **sashrelief:** This option will represent the border type around each sash. Its default value is FLAT.
- **sashwidth:** This option will specify the sash width whose default value is 2 pixels.
- **showhandle:** This option will display handles when set to True. Its default value is False.

Some of the commonly used methods in this widget are as follows:

- **add(child, options):** This method will add a child window to the paned window.
- **get(startindex [,endindex]):** This method will get the text within the specified range.
- **config(options):** This method will allow the widget to configure within the specified options. A dictionary is returned containing all option values if no options were given.

We shall see some examples:

```
from tkinter import *

#main window creation
myroot = Tk()

#window size
myroot.geometry('300x300')

# 1st paned window object
mypw1 = PanedWindow(myroot)

#expand option for widgets to expand and fill for letting widgets adjust itself
mypw1.pack(fill = BOTH, expand = 1)

# entry widget creation
mye1 = Entry(mypw1, bd = 5, relief = 'groove', font = ('Calibri',12), bg = 'LightBlue')

# will add entry widget to the panedwindow
mypw1.add(mye1)

# 2nd paned window object
mypw2 = PanedWindow(mypw1, orient = VERTICAL)

#adding 2nd paned window to the 1st paned window
mypw1.add(mypw2)

# spinbox object creation
mye2 = Spinbox(mypw2, from_ = 10, to = 20, font = ('Calibri',12), bg = 'LightPink')

# another entry widget creation
mye3 = Entry(mypw2, bg = 'LightGreen',font = ('Calibri',12) )

#setting the value to 3
mye3.insert(0,3)
```

```
# to show sash
mypw1.configure(sashrelief = RAISED)

# subtract function
def subtract():
    num1 = int(mye2.get()) # getting value of spinbox
    num2 = int(mye3.get()) # getting value of entry
    mydata = str(num1-num2)
    mye1.insert(1,mydata)

# adding spinbox to the 2nd paned window
mypw2.add(mye2)

# adding entry to the 2nd paned window
mypw2.add(mye3)

# creation of button widget
mybtn = Button(mypw2, text = "Subtract", command = subtract)

# adding button to the 2nd paned window
mypw2.add(mybtn)

# infinite loop
myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.7*:

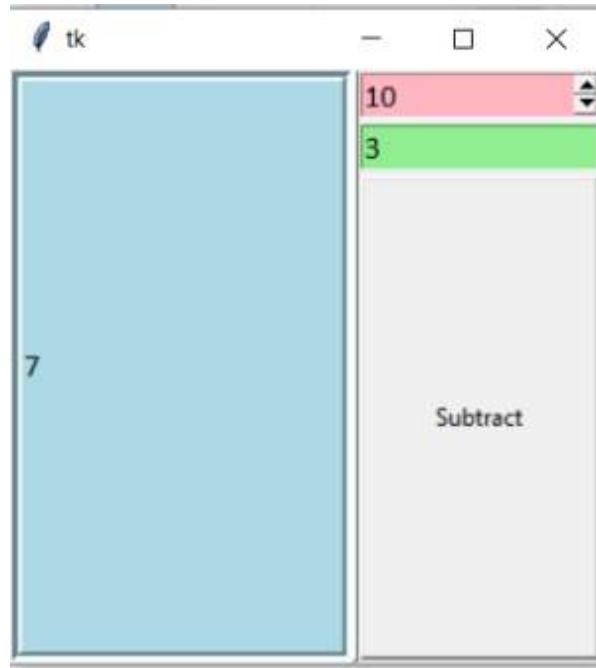


Figure 6.7: Output

Note: The preceding code is covered in Program Name: **Chap6_Example7.py**

In this code, we have created 2 panedwindows, one with default orientation HORIZONTAL and the other with orientation VERTICAL. The 2nd pane window is added to the 1st pane. The 1st pane contains an Entry widget and the 2nd pane contains 1 Spinbox widget and 1 Entry widget, followed by one button widget which will calculate the subtraction of 2 numbers and will insert the result into the Entry widget of the 1st pane. So, here we have created a 3-pane widget.

We can also increase the separator line width sash as shown:

```
from tkinter import *

#main window creation
myroot = Tk()

#window size
myroot.geometry('300x300')

#paned window object
mypw1 = PanedWindow(myroot,orient ='vertical')

#expand option for widgets to expand and fill for letting widgets adjust itself
mypw1.pack(fill = BOTH, expand = 1)

# Checkbutton object
mychk = Checkbutton(mypw1, text ="I am checkbutton")
mychk.pack(side = TOP)

# Adding Checkbutton to panedwindow
mypw1.add(mychk)

# Radiobutton object
myr1 = Radiobutton(mypw1, text ="I am radiobutton")
myr1.pack(side = TOP)

# Adding Radiobutton to panedwindow
mypw1.add(myr1)
```

```

# button object
mybtn1 = Button(mypw1, text ="I am button")
mybtn1.pack(side = TOP)

# Adding button to panedwindow
mypw1.add(mybtn1)

# Tkinter string variable
mystr = StringVar()

# entry widget
mye1 = Entry(mypw1, textvariable = mystr, font =('arial', 15, 'bold'))
mye1.pack()

# will focus on entry widget particularly
mye1.focus_force()

mystr.set('          PanedWindow')

# Will show sash
mypw1.configure(sashrelief = RAISED, sashwidth = 5)

# adding entry widget to the paned window
mypw1.add(mye1)

#infinite loop
myroot.mainloop()

```

Output:

The output can be seen in *Figure 6.8*:



Figure 6.8: Output

Note: The preceding code is covered in Program Name: Chap6_Example8.py

We can also increase the padding around each sash by using the `sashpad` option, as shown:

```
# Will show sash
mypw1.configure(sashrelief = RAISED, sashwidth = 5, sashpad = 5)
```

Just observe the difference, as shown in [Figure 6.9](#):

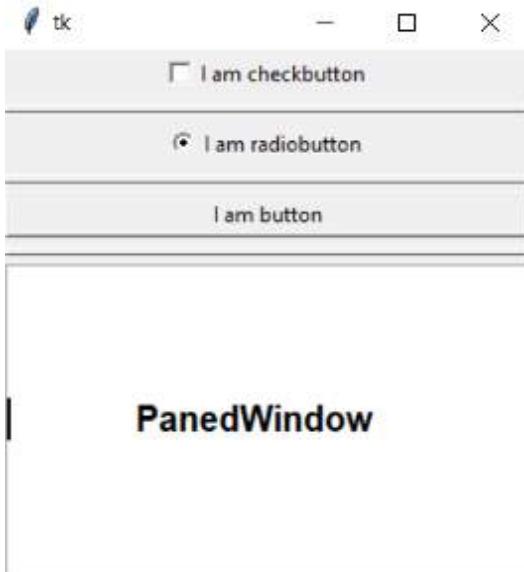


Figure 6.9: Output after increasing the padding

tkinter Toplevel widget

This widget is like Frame which is always contained in a new window, that will first create and then display the toplevel windows. These windows will be managed directly by the Windows manager. This widget may or may not have the parent window on top of them. This widget will be required whenever we want to see some group of widgets on the new window or we want to display some extra information and so on.

The syntax is as follows:

```
mytoplevel1= Toplevel(options...)
```

Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, cursor, font, class_, fg, height, relief, and width.

We have seen most of the options but some undiscussed options are as follows:

- **class_**: In this option, when we select the text within a text widget, and the text selected in the text manager will be exported. If set to 0, this option's behavior is avoided.

Some of the methods associated with this widget are as follows:

- **deiconify()**: This method is used to display the window.
- **iconify()**: This method will convert the top-level window into an icon without destroying it.
- **frame()**: This method will show a system-dependent window identifier.
- **group(window)**: This method will add a window to a specified window group.
- **state()**: This method will get the current window state, which could be normal, iconic, zoomed, and withdrawn.
- **protocol(name, function)**: This method will mention a function that will be called for the specific protocol.
- **transient([window])**: This method will convert the window to a temporary window for the given master, when there is no argument.
- **withdraw()**: This method will remove the window from the screen but it will not be destroyed.
- **maxsize(width, height)**: This method will define the maximum size for the window.
- **minsize(width, height)**: This method will define the minimum size for the window.
- **resizable(width, height)**: This method will help to control the window resizing and check whether it can be resizable or not.
- **positionfrom(who)**: This method will define the position controller.
- **sizefrom(who)**: This method will define the size controller.
- **title(string)**: This method will define the window title.

Let us see a basic example of this widget:

```
from tkinter import *

myroot = Tk()
myroot.geometry("250x250")

def mynavigate():
    # top level object for creation of a new window
    mytopobj = Toplevel(myroot)
    mytopobj.geometry('250x250')
    #getting the title for the window
    mytopobj.title('NewWindow')
    # infinitely running mainloop
    mytopobj.mainloop()

# button object for opening of new window on button click
mybtn1 = Button(myroot, text = "Mynavigate", command = mynavigate)
# positioning the button
mybtn1.place(x=100,y=100)

# infinitely running mainloop
myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.10*:

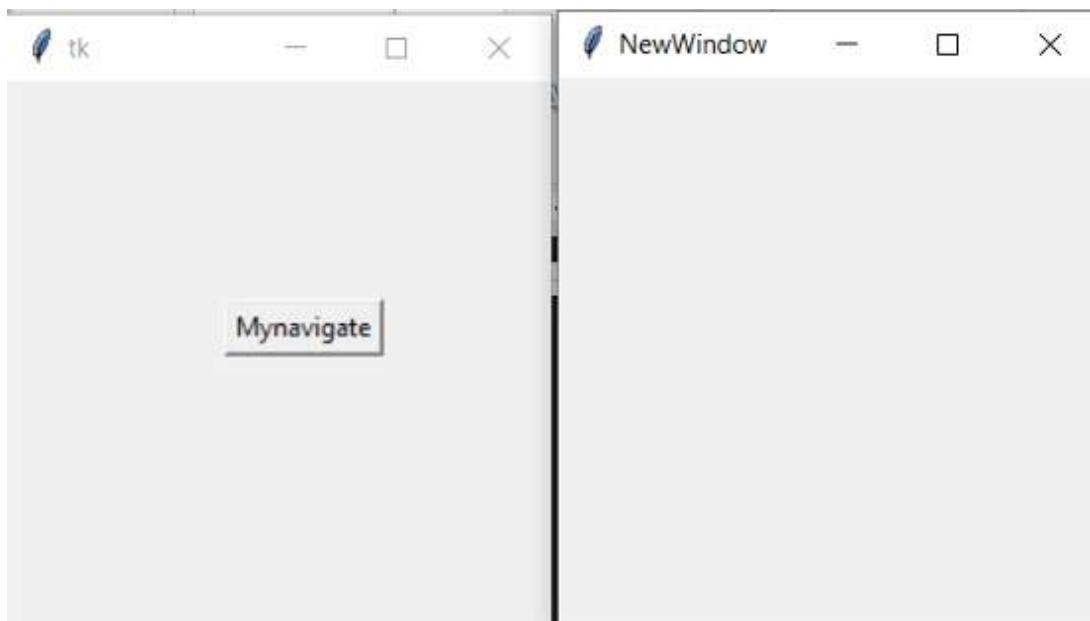


Figure 6.10: Output

Note: The preceding code is covered in Program Name: Chap6_Example9.py

In this code, on clicking the **Mynavigate** button, a new top-level window is opened, having all the properties that a main window should have.

An important point to note is that when the empty window is created, it is 200x200 pixels.

We can create multiple toplevels over one another, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry("250x250")

def mynavigate():
    # top level object for creation of a new window
    mytopobj1 = Toplevel(myroot)
    mytopobj1.geometry('250x250')
    #getting the title for the window
    mytopobj1.title('MyToplevel1')

    myl1 = Label(mytopobj1, text = 'This is a toplevel1 window')
    myl1.pack(pady = 10)
```

```
mybtn1 = Button(mytopobj1, text = 'MyToplevel2 window', command = func_myplevel2)
mybtn1.pack(pady = 10)

mybtn2 = Button(mytopobj1, text = 'Exit', command = mytopobj1.destroy)
mybtn2.pack(pady = 10)

# infinitely running mainloop
mytopobj1.mainloop()

def func_myplevel2():
    # top level object for creation of a new window
    mytopobj2 = Toplevel(myroot)
    mytopobj2.geometry('250x250')
    #getting the title for the window
    mytopobj2.title('MyToplevel2')

    myl1 = Label(mytopobj2, text = 'This is a toplevel2 window')
    myl1.pack(pady = 10)

    mybtn2 = Button(mytopobj2, text = 'Exit2', command = mytopobj2.destroy)
    mybtn2.pack(pady = 10)

    # infinitely running mainloop
    mytopobj2.mainloop()

# button object for opening of new window on button click
mybtn1 = Button(myroot, text = "MyToplevel1", command = mynavigate)
# positioning the button
mybtn1.place(x=100,y=100)

# infinitely running mainloop
myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.11*:



Figure 6.11: Output

Note: The preceding code is covered in Program Name: Chap6_Example10.py

In this code, we are trying to create multiple top levels over one another. When the MyToplevel1 button is clicked on the main window (A), a new toplevel1 window titled MyToplevel1 window is created (B). It contains a button MyToplevel2 window and an Exit button.

On clicking the MyToplevel2 window button, the user will navigate to the next MyToplevel2 window. However, when the **Exit** button is clicked, then the MyToplevel1 window will be closed. Similarly, on clicking the **Exit2** button, the MyToplevel2 window will be closed. **Please note that we have positioned the GUI forms for you for better understanding. When you will run the program, you just need to observe how the output will come.**

We can control how the windows are stacked on each other by using the **lift()** method to change the order:

```
from tkinter import *

myroot = Tk()
myroot.geometry("300x300")
myroot.title('Main window')
mytopobj = Toplevel(myroot)

mytopobj.title('New window')

mytopobj.geometry("300x300")
myroot.lift(mytoplevel)

# infinitely running mainloop
myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.12*:

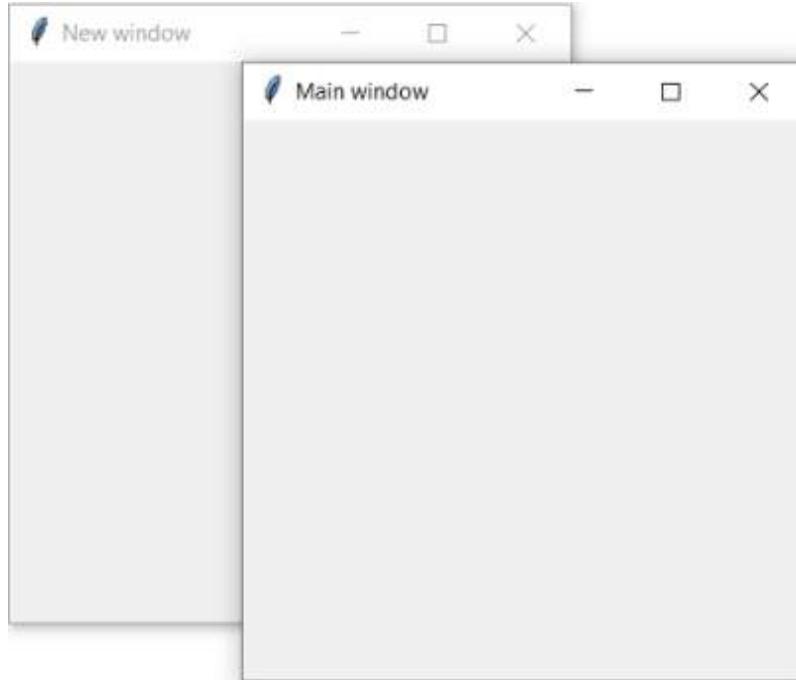


Figure 6.12: Output

Note: The preceding code is covered in Program Name: Chap6_Example11.py

We can see that the focus is on the Main window first, which is in front of the New window, that is, the position is lifted. **Just run the code yourself and observe the output. The GUI position shown next is for your better understanding.**

We can control whether the window is visible or not by changing the state using the state method. The default state is normal. We can make the window maximized by setting its state to zoom.

```
from tkinter import *

myroot = Tk()

myroot.geometry("300x300")
myroot.title('Main window')
mytopobj = Toplevel(myroot)

mytopobj.title('New window')

mytopobj.geometry("300x300")
mytopobj.lift(myroot)
mytopobj.state('zoomed')

# infinitely running mainloop
myroot.mainloop()
```

Output:

The output can be seen in *Figure 6.13*:

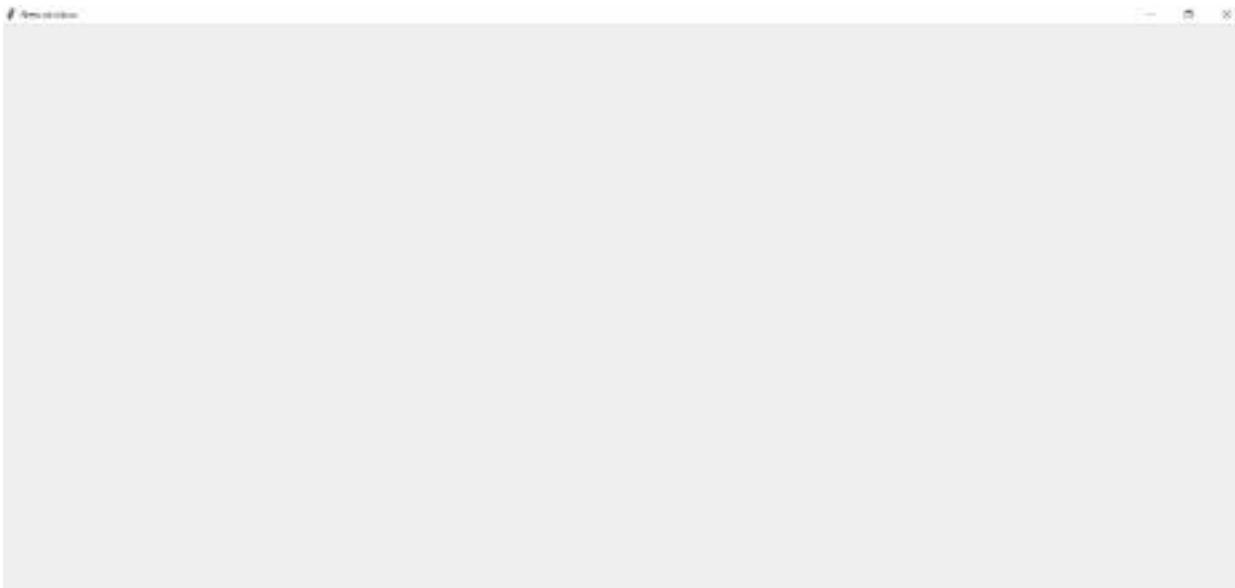


Figure 6.13: Output

Note: The preceding code is covered in Program Name: Chap6_Example12.py

The New window is expanded with the geometry size to fit the entire screen. The Main window screen will be hidden behind the above form.

We can hide the window by setting its state to be withdrawn, as shown:

```
mytopobj.state('withdrawn')
```

Refer to [*Figure 6.14:*](#)

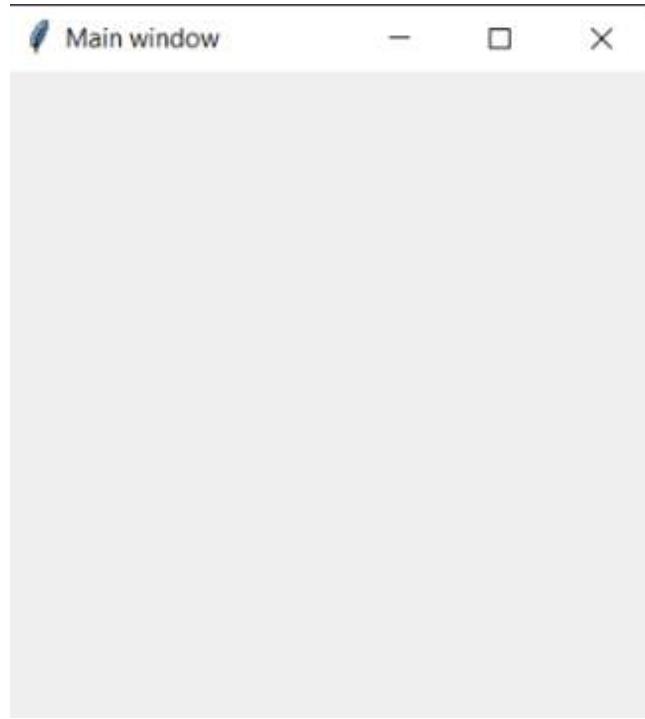


Figure 6.14: View after hiding the window

We can only see the Main window as the new window is hidden from the taskbar.

If we want to minimize the window so that we can access it from the taskbar, we can use the iconic state, as shown:

```
mytopobj.state('iconic')
```

Refer to [Figure 6.15](#):

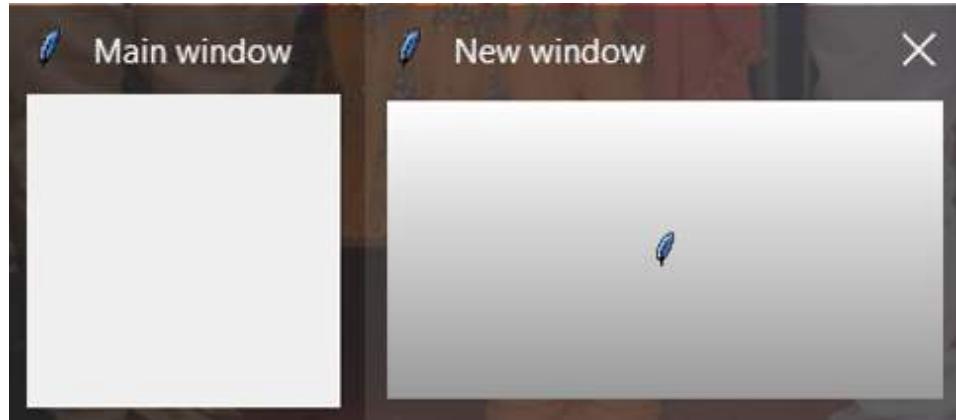


Figure 6.15: View after minimizing the window

There are some shortcut methods that can be switched between iconic and normal states. We can get the same output shown in the taskbar by using **iconify()** method. The code is as follows:

```
from tkinter import *

myroot = Tk()
myroot.geometry("300x300")
myroot.title('Main window')
mytopobj = Toplevel(myroot)

mytopobj.title('New window')

mytopobj.geometry("300x300")
mytopobj.lift(myroot)
mytopobj.iconify()

# infinitely running mainloop
myroot.mainloop()
```

**Note: The preceding code is covered in Program Name:
Chap6_Example12_2.py**

It will return to its normal state by using **deiconify()** method, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry("300x300")
myroot.title('Main window')
mytopobj = Toplevel(myroot)

mytopobj.title('New window')

mytopobj.geometry("400x300+50+100")
mytopobj.lift(myroot)
mytopobj.deiconify()

# infinitely running mainloop
myroot.mainloop()
```

We will get the normal output, as shown in the following *Figure 6.16*:

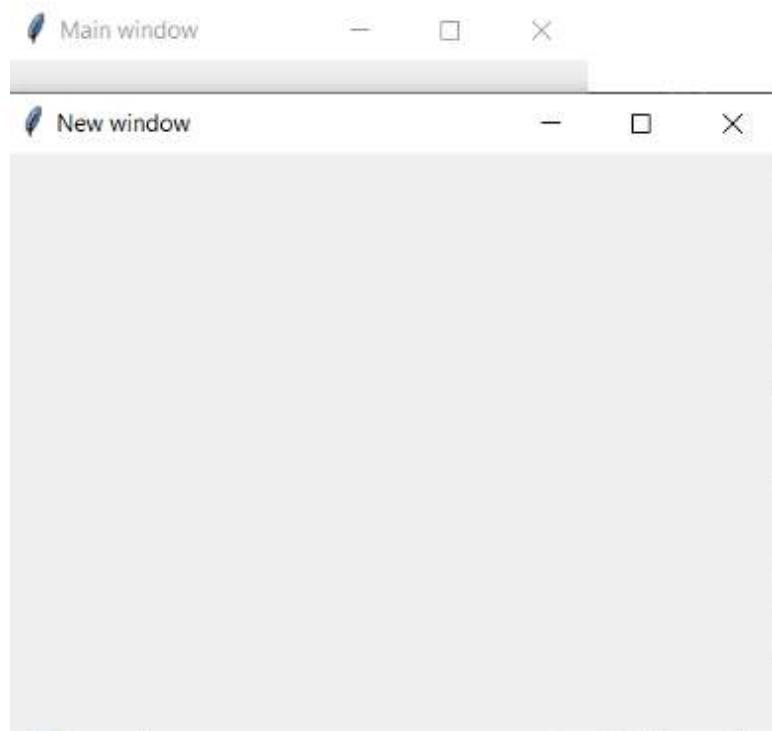


Figure 6.16: Output

Note: The preceding code is covered in Program Name: Chap6_Example13.py

Here, we have not positioned the output and we have got the desired result because we have shifted the window 50 pixels from the top-left corner to the right, and 100 pixels down from the top-left screen corner.

We can restrict the window size by using the **maxsize** and **minsize** methods, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry("300x300")
myroot.title('Main window')
mytopobj = Toplevel(myroot)

mytopobj.title('New window')

mytopobj.geometry("400x400+50+100")
mytopobj.lift(myroot)
mytopobj.maxsize(400,400)
mytopobj.minsize(200,200)

# infinitely running mainloop
myroot.mainloop()
```

Output when resized to the minimum window:

Refer to *Figure 6.17*:

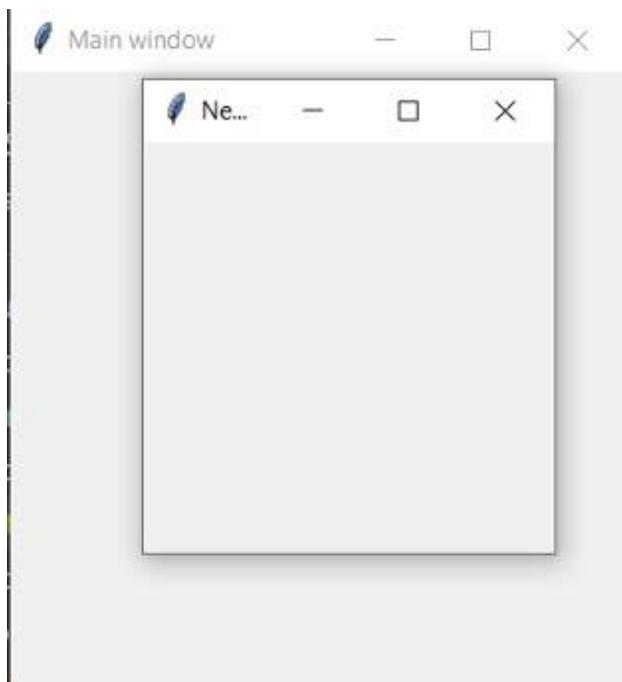


Figure 6.17: Output

Output when resized to the maximum window:

Refer to *Figure 6.18*:

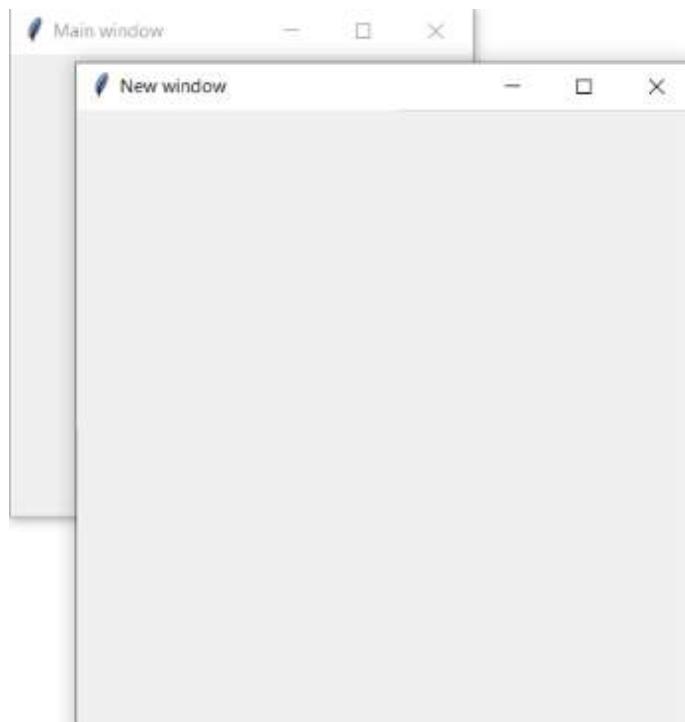


Figure 6.18: Output

Note: The preceding code is covered in Program Name: Chap6_Example14.py

Note: Here, we have adjusted the size and position of the window form, for your better understanding.

If we will be using the destroy method, then all the child windows will also be deleted. So, on using destroy method on the myroot window, the New window will be deleted. However, when we will be using destroy method on the New window, then only the above window will be deleted. However, the Main window will remain intact.

Conclusion

In this chapter, we have learned how to set positions of different widgets in a Frame widget, along with the provision of padding. Then we looked into the variant of the Frame widget which is tkinter LabelFrame where users were able to see frame features along with label display. Then we saw creation of a tabbed widget with the help of tkinter Notebook widget where the reader can create multiple tabs, which can be attached with clickable buttons. The importance of tkinter PanedWindow widget was well explored which was containing horizontal or vertical stacks of child widgets. Finally, we looked into tkinter Toplevel widget with crystal clear concepts explanation for the creation and display of top-level windows.

Points of remember

- Use frames to assemble related widgets. The code may become better organized and simpler to read as a result.
- Label frames with the help of labels. The code may become more readable and understandable as an outcome.
- Create tabbed user interfaces with notebooks. This might be a great approach for arranging a lot of information in a constrained area.
- Resizable panes can be created via paned windows. This can be a great approach for developing layouts that can be adjusted for

different sizes of screens.

- To make new windows, use tkinter Toplevel widgets. Create dialog boxes, message boxes, and other types of windows using this approach.

Questions

1. Explain the tkinter Frame widget in detail.
2. Which widget is used to arrange the widget's position? Explain in detail with a suitable example.
3. Write a program to create an entry panel for age, name, and gender using the tkinter Frame widget.
4. Which widget is used for grouping the number of interrelated widgets as well as for drawing a border around its child widgets? Explain with a suitable program.
5. Explain the tkinter LabelFrame Widget and its syntax.
6. Explain the tkinter Tabbed/Notebook Widget with its syntax.
7. Explain the tkinter PanedWindow Widget in detail.
8. Which widget is a container widget that contains any number of child widgets (panes) and can be arranged either vertically or horizontally? Explain with a suitable program.
9. Explain the tkinter Toplevel Widget with its syntax and usage in GUI application building.
10. Which widget is similar to the Frame widget and always contained in a new window and will first create and then display the toplevel windows.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 7

Getting Insights of Item Widgets in tkinter

Introduction

Sometimes, there is a requirement to allow users to select one or more than one item from a list. We may require an application for menu creation, which either allows users to select different options or requirement for displaying a list of items, or to filter and sort a list of items based on name, date, hobby and so on. We do require tkinter Listbox widget as it is easy to use, is powerful and can be customizable as per our needs.

Structure

In this chapter, we will discuss the following topic:

- tkinter Listbox widget

Objectives

After going through this chapter, the reader will learn about the tkinter Listbox widget where the user can display different types of lists of items and a number of items can be selected from the list. Different select mode examples will be viewed along with the scrollbar attached to this widget.

tkinter Listbox widget

This widget will display the item list to the user. Only text items can be placed in the Listbox and the same font and color will be present in the text

items. A user has the option to choose one or more items from the list depending on the configuration.

The syntax is as follows:

```
mylb1= Listbox(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bd, bg, cursor, font, height, fg, highlightcolor, highlightthickness, selectbackground, relief, selectmode, xscrollcommand, yscrollcommand, and width.

We have seen most of the options but some undiscussed options are as follows:

- **highlightthickness**: This option will represent the focus highlight thickness.
- **selectmode**: This option will specify the number of items that can be chosen from the list and can be set to different modes such as:
 - **SINGLE**: In this mode, we can select only one line and the mouse cannot be dragged. The line is selected whenever we click the button1.
 - **BROWSE**: It is the default mode where the user can select only one line out of a listbox. If an item is clicked and the mouse is dragged to a different line, then the item selected will follow the mouse.
 - **EXTENDED**: In this mode, the adjacent group of lines can be selected at once by clicking on the first line and dragging it to the last line.
 - **MULTIPLE**: In this mode, any number of lines can be selected at once. If any line is clicked, it will be toggled whether or not it is selected.

- **xscrollcommand**: This option will link the listbox widget to the horizontal scrollbar so that the user can scroll the listbox horizontally.
- **yscrollcommand**: This option will link the listbox widget to the vertical scrollbar so that the user can scroll the listbox vertically.

Some of the useful methods are as follows:

- **activate(index)**: This method will select the lines specified by the given index.
- **curselection()**: This method will return an empty tuple if nothing is selected. However, it will return a tuple having the line numbers of the selected elements whose counting is from 0.
- **delete(first, Last = None)**: This method will delete the lines whose indices are in the range [first, last].
- **get(first, Last = None)**: This method will return a tuple containing the text of the line whose indices are in the range [first, last].
- **index(i)**: This method will place the line at the specified index at the widget's top.
- **insert(index, *elements)**: This method will allow inserting one or more lines in the above widget before the line specified by the index. If we want to add new lines to the end of this widget, then use END as the first argument.
- **nearest(y)**: This method will return the nearest line index to the y coordinate of the above widget.
- **size()**: This method will return the number of lines that are present in the above widget.
- **see(index)**: This method will adjust the above widget's position so that the line referred to by the index, is visible.
- **xview()**: This method will make the widget horizontally scrollable.
- **xview_moveto(fraction)**: This method will make the widget horizontally scrollable by the fraction of the width of the longest line which is present in the widget.

- **xview_scroll(number, what):** This method will make the widget horizontally scrollable by the number of characters specified. The argument can use either UNITS or PAGES to scroll by characters or by pages (widget width).
- **yview():** This method will make the widget vertically scrollable.
- **yview_moveto(fraction):** This method will make the widget vertically scrollable by the fraction of the width of the longest line which is present in the widget.
- **yview_scroll(number, what):** This method will make the widget vertically scrollable by the number of characters specified.

We shall see some examples for more clarification of the above widget usage. Let us see how to create a simple listbox, using the following code:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('350x350')
myroot.title('My ListBox')

def myget():
    mylinenumber = mylb1.curselection() # getting the line number
    for loop in mylinenumber:
        print(loop, ':', mylb1.get(loop)) # getting the item of that line number

# creation of listbox with specified width and height
mylb1 = Listbox(myroot, width = 30, height = 15, bg = 'Light-Green', font = ('Verdana',12)) # default width = 20 no. of characters in one line and height = 10 (no. of lines)
# insertion of one or more lines into the listbox specified by index
mylb1.insert(1,'Hindi')
mylb1.insert(2,'English')
mylb1.insert(3,'Telugu')

mybtn = Button(myroot, text = 'Line number display', command = myget)
mybtn.pack()

myroot.mainloop() # display window until we press the close button
```

**Output:
button is clicked:**

Output when

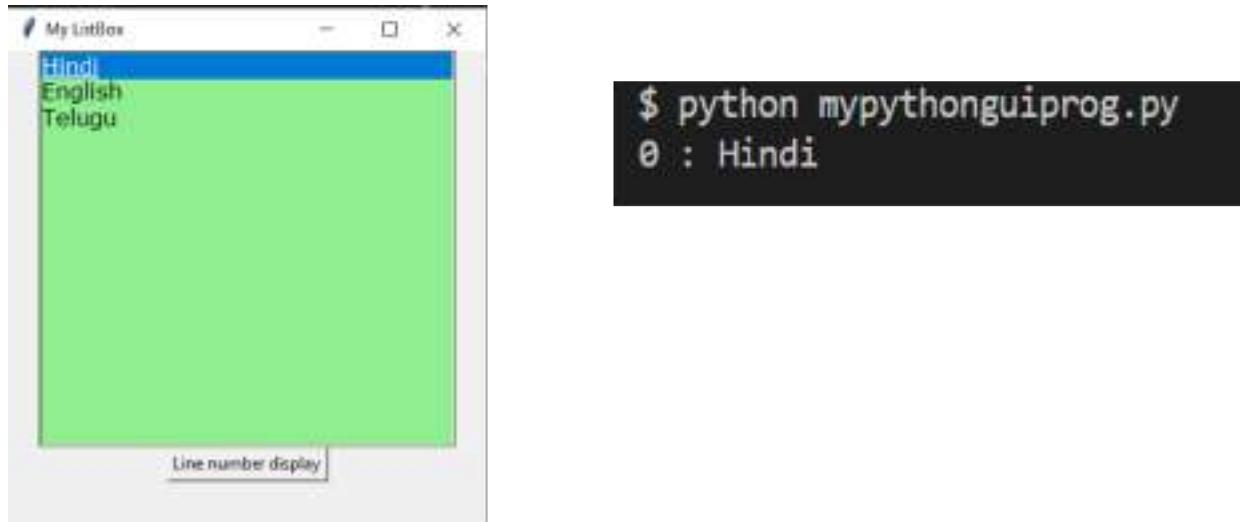


Figure 7.1: Output

Note: The preceding code is covered in Program Name: Chap7_Example1.py

Similarly, when English and Telugu are selected, and the line number display is clicked, we will get the following output:

```
$ python mypythonguiproj.py
1 : English
2 : Telugu
```

Figure 7.2: Output

In the above code, the selected mode is BROWSE by default. There are 3 items in a listbox and when selected, the output shown in [Figure 7.2](#) will be displayed.

Now, we shall see the selectmode in the listbox:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('350x350')
myroot.title('My ListBox BROWSE MODE')

# creation of listbox with specified width and height
mylb1 = Listbox(myroot, width = 30, height = 15, font = ('Ver-
dana',12), selectmode = BROWSE) # default width = 20 no. of charac-
ters in one line and height = 10 (no. of lines)
# insertion of one or more lines into the listbox specified by index
mylb1.insert(1,'Hindi')
mylb1.insert(2,'English')
mylb1.insert(3,'Telugu')
mylb1.insert(4,'Tamil')
mylb1.pack()

myroot.mainloop() # display window until we press the close button
```

Output:

The output is shown in *Figure 7.3*:



Figure 7.3: Output

Note: The preceding code is covered in Program Name: Chap7_Example2.py

Just press the left click or button 1, and drag the mouse to where we can see that the selection will follow the mouse.

In the same example, if we change the selection mode to SINGLE, then we can select only a line and we cannot drag the mouse. Refer to [*Figure 7.4:*](#)

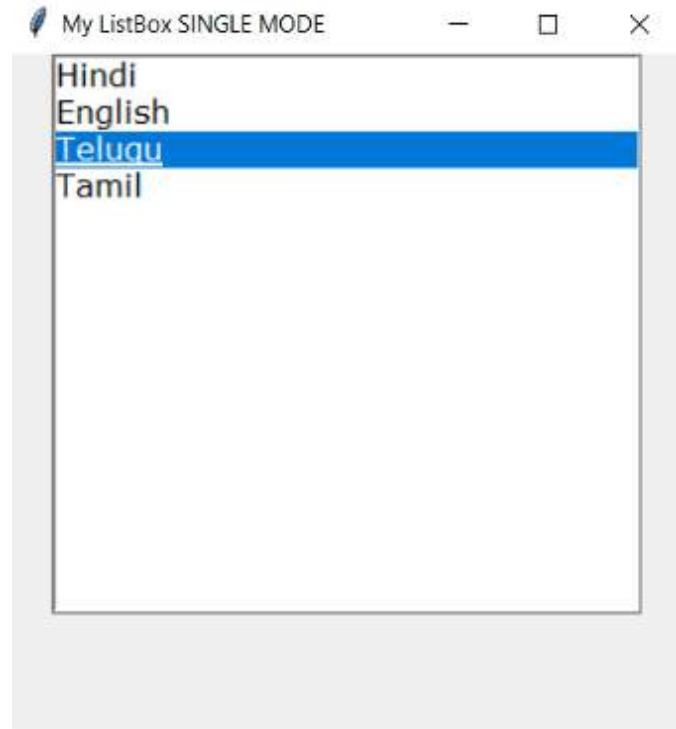


Figure 7.4: Output when selectmode is SINGLE

If we change the selectmode to MULTIPLE, then we can select any number of lines at once. If any line is clicked, it will be toggled whether or not it is selected. Refer to the following [Figure 7.5](#):

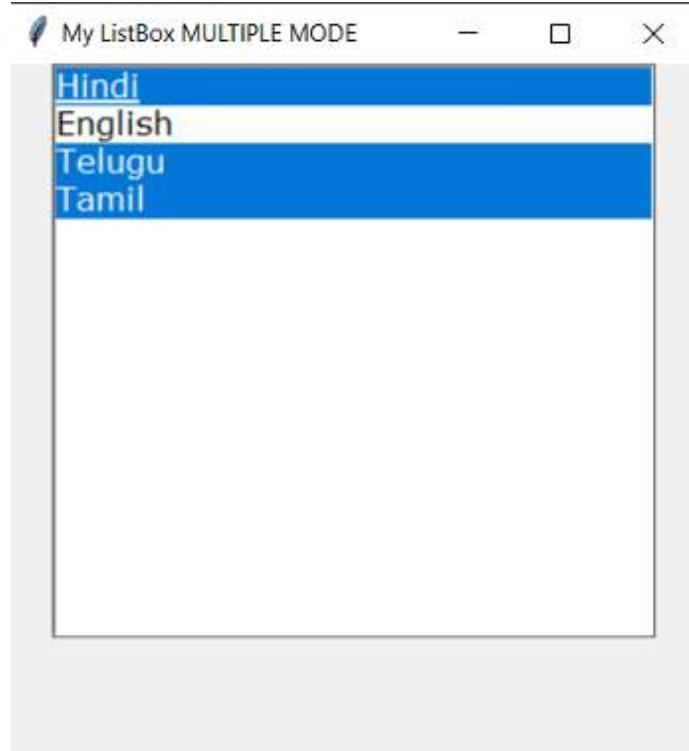


Figure 7.5: Output when selectmode is MULTIPLE

If we change the selectmode to EXTENDED, then adjacent lines are selected at once by clicking on the first line and dragging to the last line, as shown in [Figure 7.6](#):



Figure 7.6: Output when selectmode is EXTENDED

We can delete the active item from the list as shown:

```

from tkinter import * # importing module

myroot = Tk() # window creation and initialize the interpreter
myroot.geometry('350x350')
myroot.title('My ListBox delete')

# creation of listbox with specified width and height
mylb1 = Listbox(myroot, width = 30, height = 15, font = ('Ver-
dana',12)) # default width = 20 no. of charac-
ters in one line and height = 10 (no. of lines)
# insertion of one or more lines into the listbox specified by index
mylb1.insert(1,'Hindi')
mylb1.insert(2,'English')
mylb1.insert(3,'Telugu')
mylb1.insert(4,'Tamil')
mylb1.pack()

mybtn1 = Button(myroot, text = 'Mydelete', command = lambda mylb1=m-
ylb1: mylb1.delete(ANCHOR))
mybtn1.pack()

def mysize():
    print(mylb1.size())

# the item selected will be deleted from the listbox
mybtn2 = Button(myroot, text = 'Mysize', command = mysize )
mybtn2.pack()

myroot.mainloop() # display window until we press the close button

```

Output before deletion:

When the **Mysize** button is clicked, the number of lines in the listbox is 4, as shown in *Figure 7.7*:

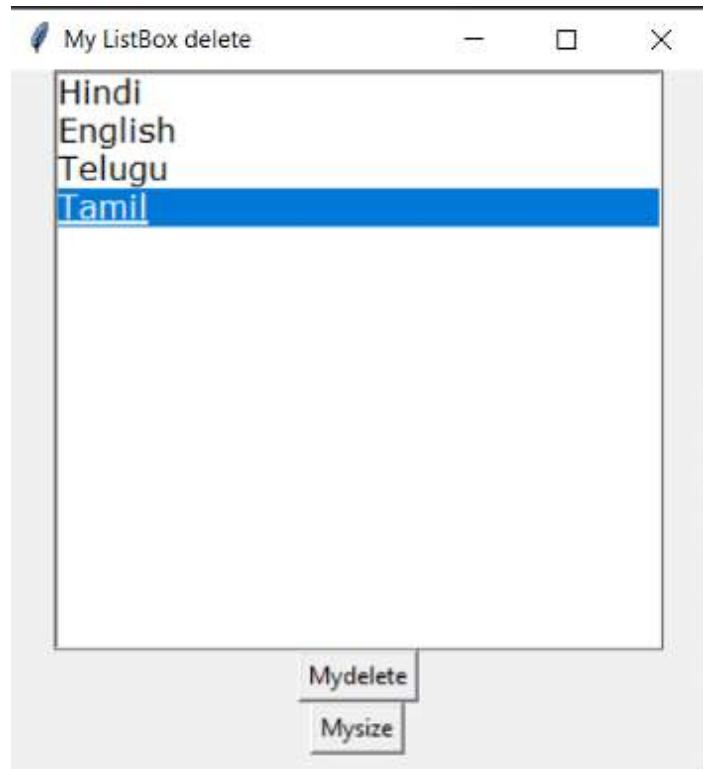


Figure 7.7: Output

Note: The preceding code is covered in Program Name: Chap7_Example3.py

In [Figure 7.7](#), we have selected the Tamil item from the listbox, and it is deleted. The number of lines in the listbox is 3 now, as shown in the following [Figure 7.8](#):

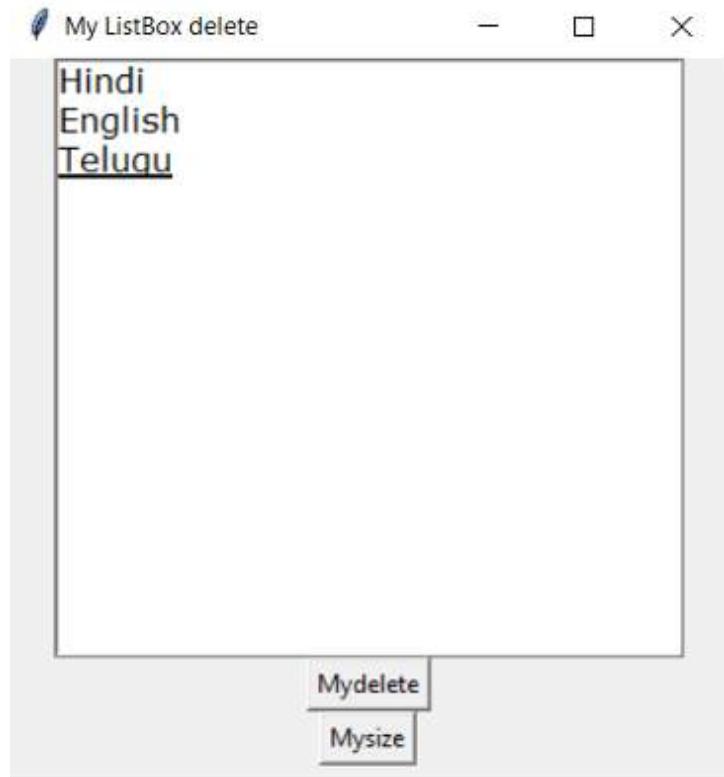


Figure 7.8: Output after deletion

We can also display vertical and horizontal scrollbar in the listbox, as shown:

```
from tkinter import *

class Scrollbar_Listbox(Tk):
    def __init__(self):
```

```
super().__init__()  
self.title('V AND H SCROLLBARS')  
  
self.mysclbar = Scrollbar(self)# scrollbar creation and at-  
taching to the main window  
self.mysclbar.pack(side=RIGHT, fill=Y) # scrollbar add-  
ed to the window right side  
  
self.sclhbar = Scrollbar(self,orient = HORIZONTAL)  
self.sclhbar.pack(side = BOTTOM,fill = X)  
  
self.mylistbox = Listbox(self,  
                         height = 600,  
                         yscrollcommand=self.mysclbar.set,  
                         xscrollcommand=self.sclhbar.set) # cre-  
ation of listbox and both horizontal and vertical scrollbars are at-  
tached to the listbox  
  
self.mylistbox.pack(expand = 1, fill=BOTH)  
  
# horizontal elements  
for loop in range(26): # insertele-  
ments from 0 to 49 in the listbox  
    self.mylistbox.insert(END, 'The element is star-  
ting from line number ' + str(loop) + ' and when multi-  
plied by 10 is: ' + str(loop*10))  
# vertical elements  
for loop in range(50): # insertele-  
ments from 0 to 49 in the listbox  
    self.mylistbox.insert(END, str(loop) + '\n')  
  
self.sclhbar.config(command=self.mylistbox.  
xview)# for need of horizontal view settings scrollbar command op-  
tion to listbox.xview method  
self.mysclbar.config(command=self.mylistbox.  
yview) # for need of vertical view settings scrollbar command op-  
tion to listbox.yview method
```

```
if __name__ == '__main__':
    myroot = Scrollbar_Listbox() # creating an instance of Scroll-
                                # bar_Listbox
    myroot.geometry('300x500')
    myroot.mainloop() # infinite loop to run the application
```

Output:

Refer to the following *Figure 7.9*:

```
V AND H SCROLLBARS - X
The element is staring from line number 0 and when
The element is staring from line number 1 and when
The element is staring from line number 2 and when
The element is staring from line number 3 and when
The element is staring from line number 4 and when
The element is staring from line number 5 and when
The element is staring from line number 6 and when
The element is staring from line number 7 and when
The element is staring from line number 8 and when
The element is staring from line number 9 and when
The element is staring from line number 10 and wher
The element is staring from line number 11 and wher
The element is staring from line number 12 and wher
The element is staring from line number 13 and wher
The element is staring from line number 14 and wher
The element is staring from line number 15 and wher
The element is staring from line number 16 and wher
The element is staring from line number 17 and wher
The element is staring from line number 18 and wher
The element is staring from line number 19 and wher
The element is staring from line number 20 and wher
The element is staring from line number 21 and wher
The element is staring from line number 22 and wher
The element is staring from line number 23 and wher
The element is staring from line number 24 and wher
The element is staring from line number 25 and wher
0
1
2
3
```

Figure 7.9: Output

Note: The preceding code is covered in Program Name: Chap7_Example4.py

Conclusion

In this chapter, we learned about the tkinter Listbox widget where we displayed different types of lists of items, and see how a number of items can be selected from the list. Different select mode examples on changing to SINGLE, MULTIPLE, EXTENDED from BROWSE were also demonstrated with examples. Finally, we saw an example of attaching both vertical and horizontal scrollbars to the above widget.

Points of remember

- We can create Listbox widget using `tk.ListBox()` constructor.
- The appearance and behavior of tkinter Listbox widget can be customized based on the number of options present.
- List of items can be displayed and one or more items can be selected by the user by using this tkinter Listbox widget, based on the application requirement.
- We can filter or sort the items in the list using this tkinter Listbox widget.
- User may choose different selectmodes such as SINGLE, EXTENDED or MULTIPLE based on application requirement.
- Different events can be triggered like `<Button-1>`, `<Double-Button-1>`, `<Double-Button-1>`, `<Double-Button-1>` and `<Double-Button-1>` based on application requirement.

Questions

1. Explain the tkinter Listbox Widget in detail.
2. Which widget will display the item list to the user? Explain in detail.
3. Write a program to create a list entry for the following cities:
 - a. Nagpur
 - b. Pune
 - c. Hyderabad

d. Mumbai

4. Which widget is used to provide a user with an option to choose one or more items from the list depending upon the configuration? Explain in detail.
5. Write short notes on Item Widgets in tkinter.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 8

Getting Insights of tkinter User Interactive Widgets

Introduction

In this chapter, we will give users the ability to design intuitive **Graphical User Interfaces (GUI)** that look visually appealing. We can construct user-friendly GUIs that give users a method to meaningfully interact with the application by using widgets such as tkinter Menu, Menubutton and Canvas Widgets. In a GUI, menus are made using the Menu widget. The usage of menus can give users access to multiple commands or options. When a button with the Menubutton widget is clicked, a menu appears. Users can frequently access additional commands or settings by using menubuttons. Users can draw or interact with things on a canvas created by the Canvas widget. Interactive visuals and video games are frequently made using canvas widget.

Structure

In this chapter, we will discuss the following topics:

- tkinter Menu widget
- tkinter Menubutton widget
- tkinter Canvas widget

Objectives

After reading this chapter, the reader will learn how to create different menus such as pop-up, top-level, and pull-down menus with the help of the tkinter Menu widget. The user can also create different applications such as Notepad, WordPad, any management software, and so on. Moreover, we will deal with the drop-down menu widget which is associated with a Menu widget called the tkinter Menubutton widget, which can display the choices when the user clicks on Menubutton. Finally, we will view the concepts of drawing different graphics like lines, rectangles, and so on, with the help of the tkinter Canvas widget.

tkinter Menu widget

This widget is a top-level menu displayed under the parent window's title bar. It provides options such as File, Edit, quit, and so on, in the application.

The syntax is as follows:

```
mymenu1= Menu(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, activeborderwidth, activebackground, activeforeground, disabledforeground, cursor, font, fg, relief, postcommand, image, selectcolor, tearoff, and title.

We have seen most of the options but some undiscussed options are as follows:

- **activeborderwidth:** This option will specify the border width of the widget when it is under the mouse.
- **disabledforeground:** This option will specify the foreground color when the state is disabled.
- **postcommand:** When the mouse is over the widget, this option can be set to any of the function.

- **tearoff**: This option will detach the menus from the main window creating floating menus and is a position from where the menu starts. When set to 1, a menu is created with dotted lines at the top and when clicked, the menu becomes floating as it will tear off the parent window. When set to 0, the menu is restricted in the main window.
- **selectcolor**: This option will show the radiobutton or checkbutton color when selected.
- **title**: This option will change the window title of the GUI application if it is set to that string.

Some of the methods used in this widget are:

- **add_command(options)**: This method will add menu items to the main menu.
- **add_checkbutton(options)**: This method will add a checkbutton to the menu.
- **add_radiobutton(options)**: This method will add a radiobutton to the menu.
- **add_separator(options)**: This method will add a separator line to the menu.
- **add_cascade(options)**: This method will create a sub-level menu to the parent menu on association of a given menu to the parent menu, where the menu items will be aligned one under the other.
- **add(type, options)**: This method will add a specific type (must be cascade, checkbutton, command, radiobutton, or separator) of the menu item to the menu.
- **delete (startindex, endindex)**: This method will delete menu items from the start index to the end index.
- **entryconfig (index, options)**: This method will modify a menu item based on the index and will change its options.
- **index(item)**: This method will return the index of the specified menu item.

- **insert_separator(index):** This method will insert a separator at the specified index.
- **invoke(index):** This method will request the menu item which we want at a specified index.
- **type(index):** This method will return the choice type which we want at a specified index, which could be either radiobutton, checkbutton, tearoff, command, cascade, or separator.

Now, we shall see some examples for better understanding:

```
from tkinter import *
```

```
myroot = Tk()
# will be called when Welcome! menu item will be clicked
def mygreet():
    print("Welcome!")

# A toplevel menu is created
mymenu = Menu(myroot)
#menu items will be added to the main menu
mymenu.add_command(label="Welcome!", command=mygreet)
mymenu.add_command(label="Quit!", command=myroot.quit) # will close the GUI application

# display of menu
myroot.config(menu=mymenu)

myroot.mainloop()
```

Output:

Figure 8.1 shows the default output:

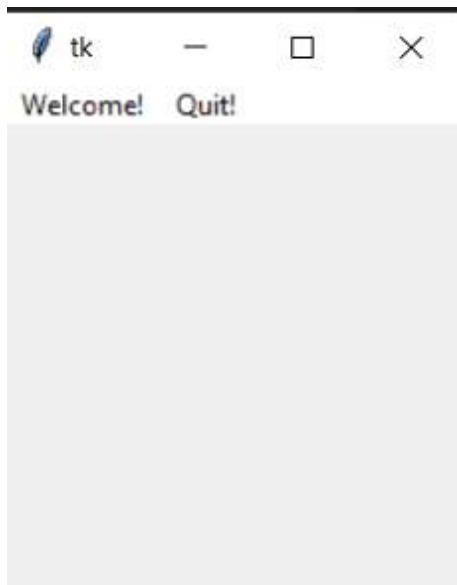


Figure 8.1: Default Output

Figure 8.2 features the output when the welcome button is clicked:

```
$ python mypythonguiproj.py
Welcome!
```

Figure 8.2: Output when the welcome button is clicked

**Note: The preceding code is covered in Program Name:
[Chap8_Example1.py](#)**

In this code, we have created a menubar and created the items Welcome! and Quit! The menu is attached to the window using `config()` method. The *Welcome!* message will be displayed on clicking the *Welcome!* menuitem. The GUI application will be closed on clicking the *Quit!* menuitem.

Now, we shall see an example where we will be adding menus such as File, and Edit Menu to the menu bar and also assign some items to these menus:

```
from tkinter import * # importing module
```

```
myroot = Tk() # window creation and initialize the
interpreter

#creating main menu
mymainmenu = Menu(myroot)
myroot.config(menu = mymainmenu) # need to attach the
above menu with the root

#creating file menu -- sub level menu corresponding to the
main menu
myfilemenu = Menu(mymainmenu, tearoff = 0) # removing the
dotted lines by setting tearoff = 0
mymainmenu.add_cascade(label = 'MyFile', menu =
myfilemenu)

# function is created to display MyNew Project Menu

def myfunc1():
    print('MyNew Project Menu')

# function is created to display MySave menu
def myfunc3():
    print('MySave Menu')

# function is created to exit the GUI application
def myfunc4():
    print('Exit')
    myroot.quit()
    exit()

#adding in the file menu
myfilemenu.add_command(label='MyNew Project', command =
myfunc1) # bind the function myfunc1 we created to the
```

```
above menuitem

myfilemenu.add_command(label='MySave', command = myfunc3)
# bind the function myfunc3 we created to the above
menuitem

myfilemenu.add_separator() # adding the separator between
MySave and MyExit

myfilemenu.add_command(label='MyExit', command = myfunc4)
# bind the function myfunc4 we created to the above
menuitem

#creating edit menu-- sub level menu corresponding to the
main menu

myeditmenu = Menu(mymainmenu)
mymainmenu.add_cascade(label = 'MyEdit', menu =
myeditmenu)

# function is created to display MyUndo menu
def myfunc2():
    print('MyUndo Menu')

# function is created to display MyCut menu
def myfunc5():
    print('MyCut Menu')

# function is created to display MyCopy menu
def myfunc6():
    print('MyCopy Menu')

# function is created to display MyRedo menu
def myfunc7():
    print('MyRedo Menu')

#adding in the file menu
```

```

myeditmenu.add_command(label='MyCut', command = myfunc5) # bind the function myfunc5 we created to the above menuitem
myeditmenu.add_command(label='MyCopy', command = myfunc6) # bind the function myfunc6 we created to the above menuitem
myeditmenu.add_command(label='MyUndo', command = myfunc2) # bind the function myfunc2 we created to the above menuitem
myeditmenu.add_command(label='MyRedo', command = myfunc7) # bind the function myfunc7 we created to the above menuitem

myroot.mainloop() # display window until we press the close button

```

Output:

Figure 8.3 shows the output with File Menu display:

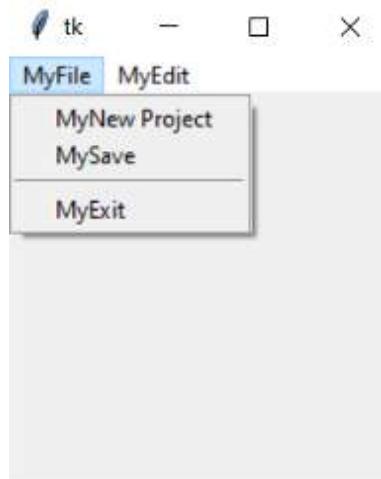


Figure 8.3: Output with File Menu display

Figure 8.4 shows the output with Edit Menu display:

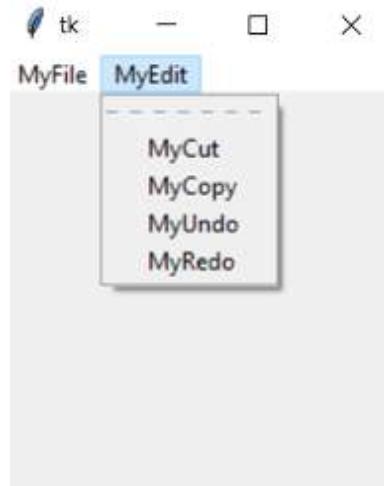


Figure 8.4: Output with Edit Menu display

Figure 8.5 shows the output when each item of File and Edit menu is clicked:

```
$ python mypythonguiprog.py
MyNew Project Menu
MySave Menu
MyCut Menu
MyCopy Menu
MyUndo Menu
MyRedo Menu
Exit
```

Figure 8.5: Output when each item of File and Edit menu is clicked

**Note: The preceding code is covered in Program Name:
Chap8_Example2.py**

We can add **radiobutton** and **checkboxbutton** to the menu and give the color when selected, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
```

```
#creating variables
mytxt_color = StringVar(myroot)
mytxt_color.set("black")

myshow = IntVar(myroot)

# creating main menu
mymenuBar = Menu(myroot)

mymenu1 = Menu(myroot) # L1

# creating submenu
mysubmenu = Menu(myroot)
mysubmenu.add_radiobutton(label="Radio 1",
variable=mytxt_color, value="black", selectcolor = 'Red')
mysubmenu.add_radiobutton(label="Radio 2",
variable=mytxt_color, value="green", selectcolor = 'Red')

mysubmenu1 = Menu(myroot)
mysubmenu1.add_checkbutton(label="Check 1",
variable=myshow, selectcolor = 'Green')

mymenuBar.add_cascade(label="MyMenu", menu=mymenu1)
mymenu1.add_cascade(label="Submenu with Radio buttons",
menu=mysubmenu)
mymenu1.add_separator()
mymenu1.add_cascade(label="Submenu with Check buttons",
menu=mysubmenu1)

myroot.config(menu=mymenuBar) # display the menu to the window

myroot.mainloop()
```

Output:

Figure 8.6 shows the default output:

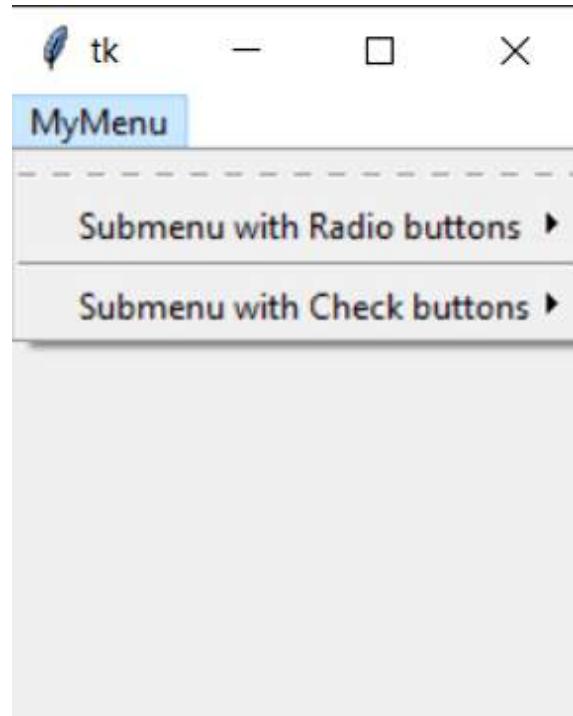


Figure 8.6: Default output

Figure 8.7 shows the output when submenus with Radio buttons is expanded. Radio 1 is checked:

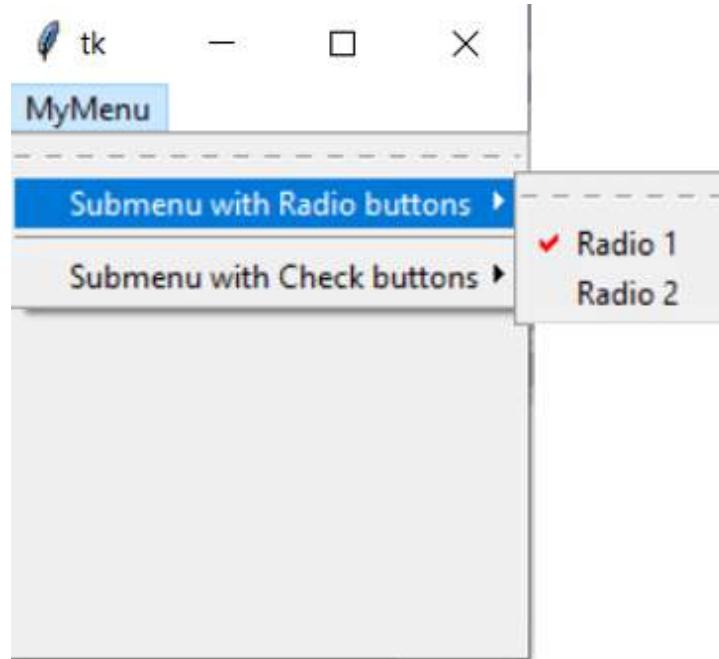


Figure 8.7: Output when submenus with Radio buttons is expanded. Radio 1 is checked

[Figure 8.8](#) shows the output when **Radio 2** is checked:

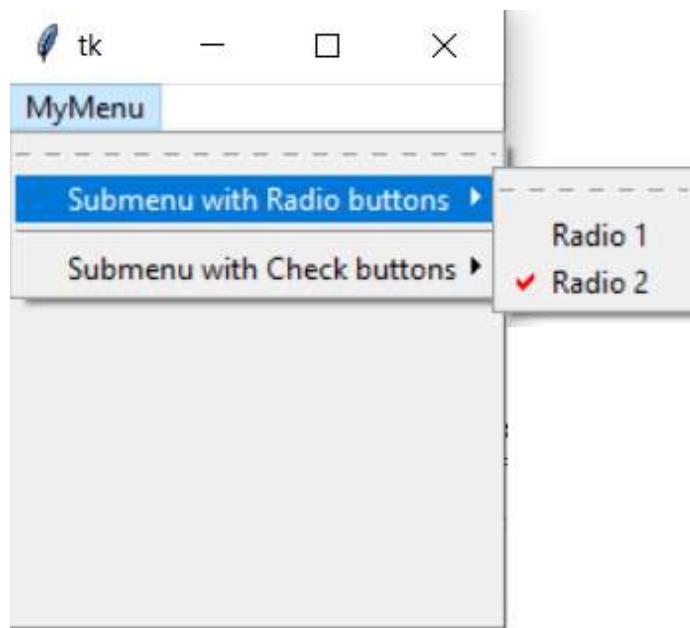


Figure 8.8: Output when Radio 2 is checked

[Figure 8.9](#) shows the output when submenus with Check buttons is expanded:

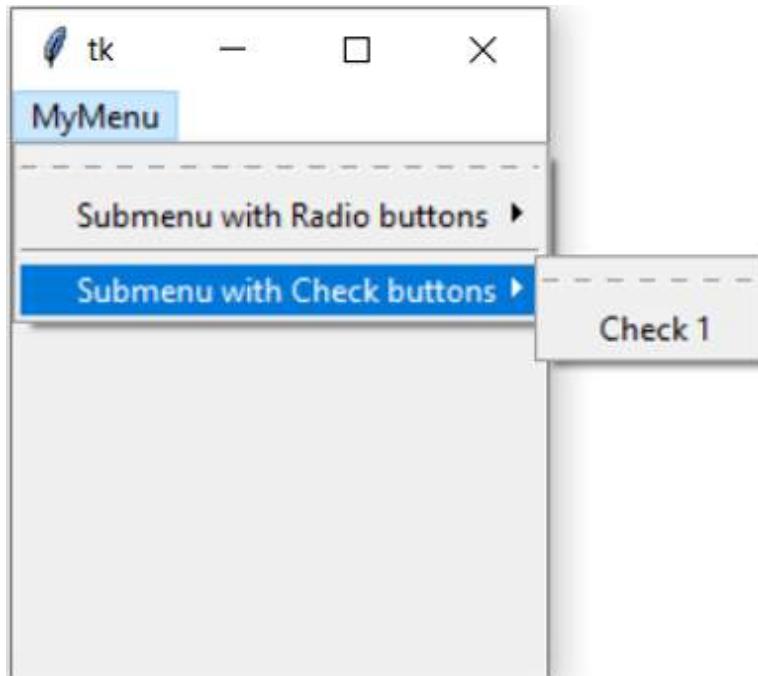


Figure 8.9: Output when submenus with Check buttons is expanded

[Figure 8.10](#) shows the output when **Check 1** is checked:

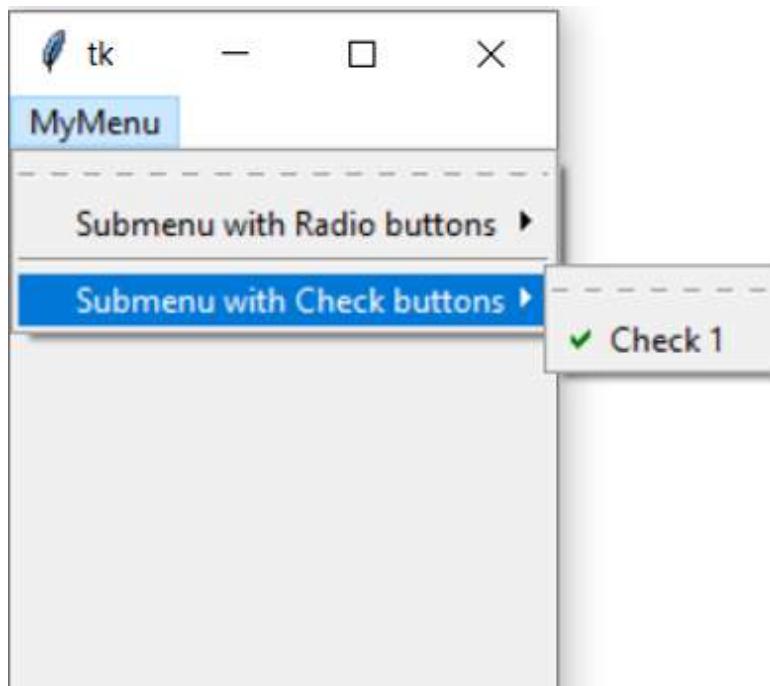


Figure 8.10: Output when Check 1 is checked

**Note: The preceding code is covered in Program Name:
Chap8_Example3.py**

In this code, we can see that we have created a menubar **Mymenu** having 2 submenus (*Figure 8.6*). The 1st submenu will contain Radiobutton menu items Radio1, and Radio2, and the 2nd submenu will contain checkbutton **Check1** menu item (*Figure 8.9*).

We can see that when one radiobutton is selected (*Figure 8.7*), the other radiobutton is unselected, and vice-versa (*Figure 8.8*). The select color of the checkbutton is Red in color. A variable is set to store which of the options are toggled and a value is set for each button. By default, we have set it to ‘black’.

Moreover, when **Check1** is clicked, then it will be selected (*Figure 8.10*). The select color of the checkbutton is Green in color. A variable is specified to hold the current checked state.

Now, suppose we click the given portion in the menu as shown in the red color rectangular region, in *Figure 8.11*:

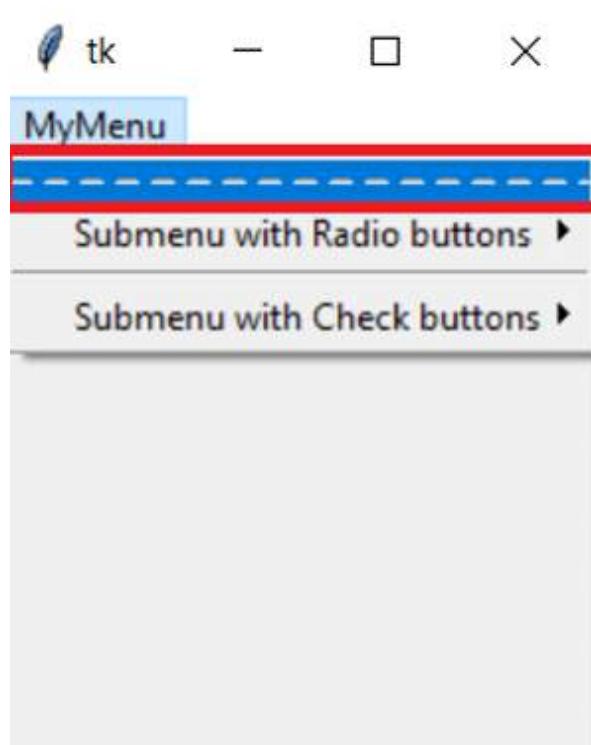


Figure 8.11: Output on detaching the dotted lines

It will detach the menus from the main window, thus creating floating menus. When set to 1, a menu is created with dotted lines at the top and when clicked, the menu becomes floating as it will tear off the parent window. Refer to [Figure 8.12](#):

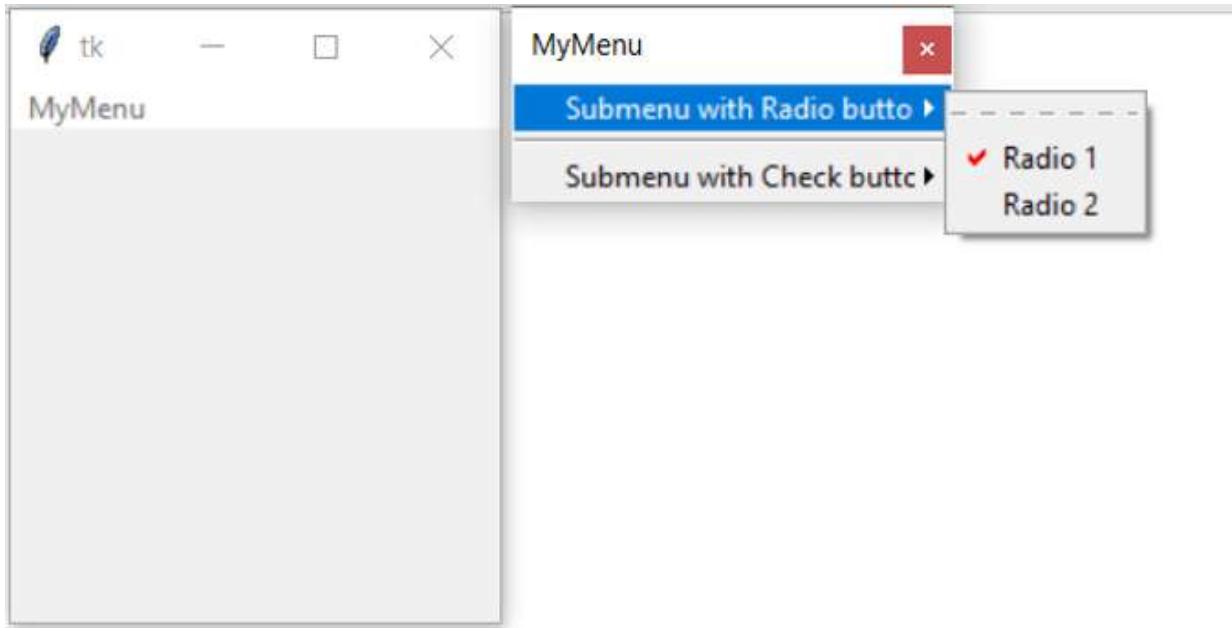


Figure 8.12: Output after detachment of dotted lines

So, in L1 in the code, we will set it to 0 so that the menu is restricted in the main window:

```
mymenu1 = Menu(myroot, tearoff = 0) # L1
```

Now, we will not see the dotted separated line, and will be restricted in the main window, as shown in [Figure 8.13](#):

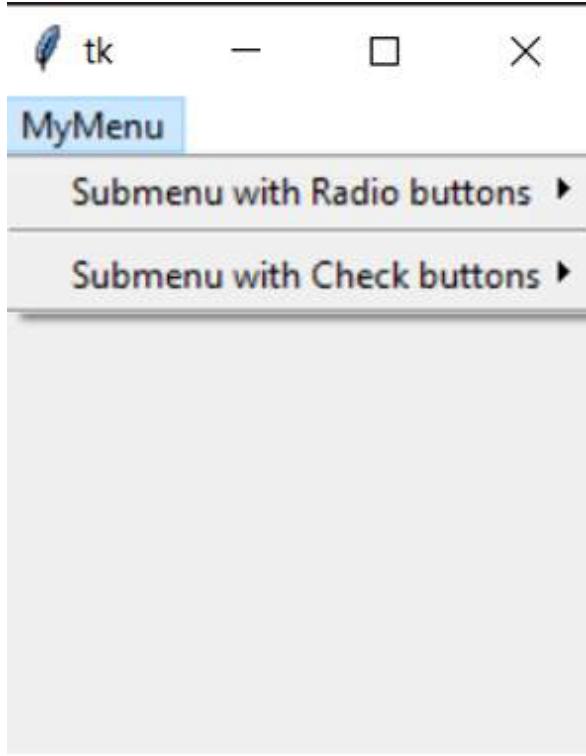


Figure 8.13: Output when `tearoff` is set to 0. No dotted line

We can also change the label of an item in the menu using `entryconfig()` method as shown:

```
from tkinter import *

myroot = Tk()
mymenu_bar = Menu(myroot)

def myselected(menu):
    menu.entryconfig(1, label="Selected!")

myedit_menu = Menu(mymenu_bar, tearoff=0)
myedit_menu.add_command(label="Demo1", command=lambda:
myselected(myedit_menu))
mymenu_bar.add_cascade(label="Edit", menu=myedit_menu)

myroot.config(menu=mymenu_bar)
```

```
myroot.mainloop()
```

Output:

Figure 8.14 shows the output before clicking **Demo1** button:

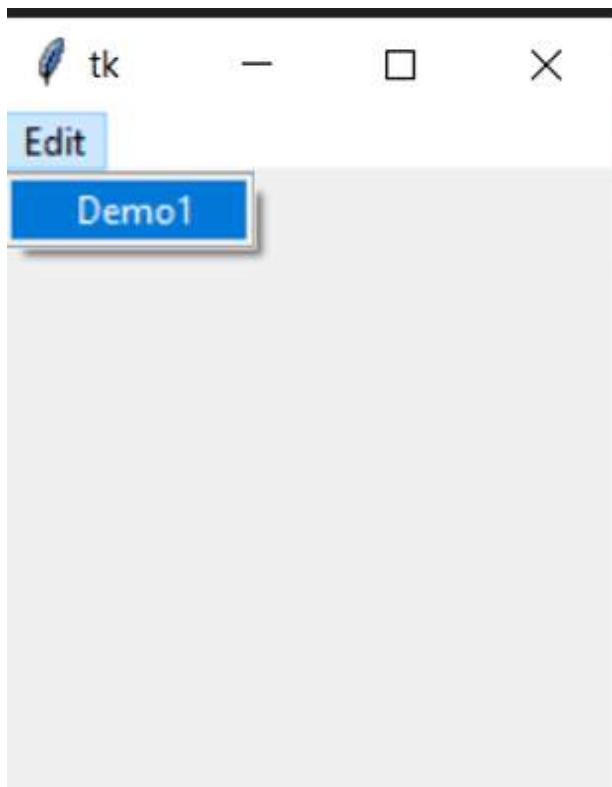


Figure 8.14: Output of before clicking Demo1 button

Figure 8.15 shows the output after clicking **Demo1** button:

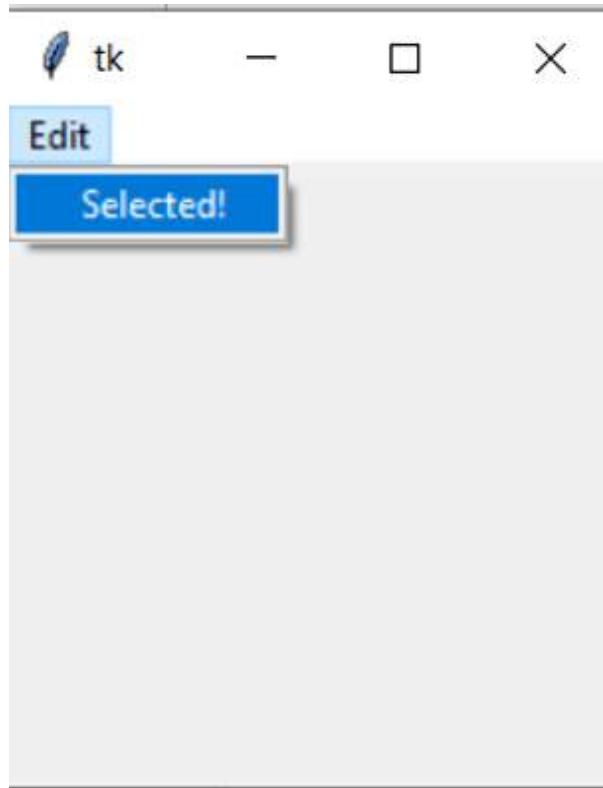


Figure 8.15: Output before clicking Demo1 Button

**Note: The preceding code is covered in Program Name:
Chap8_Example4.py**

We can add a specific type of menu item by adding it as a string to the menu. It can be a keyword argument by using the key itemType or be a positional argument.

```
from tkinter import *  
  
myroot = Tk()  
mymenu_bar = Menu(myroot)  
  
myedit_menu = Menu(mymenu_bar, tearoff=0) # L2  
myedit_menu.add_command(label="Cut")  
myedit_menu.add("command", label="Copy", command=lambda:  
print("Copy"))  
myedit_menu.add("command", label="Paste", command=lambda:
```

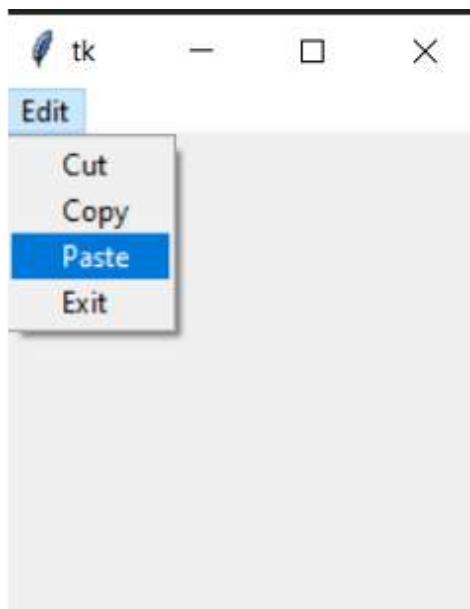
```
print("Paste")) # using
myedit_menu.add(itemType = "command", label="Exit",
command=lambda: myroot.quit()) # using itemType
mymenu_bar.add_cascade(label="Edit", menu=myedit_menu)

myroot.config(menu=mymenu_bar)

myroot.mainloop()
```

Output:

Figure 8.16 shows the output when **Paste** menu item is clicked. As we can see, **Paste** is displayed to the console:

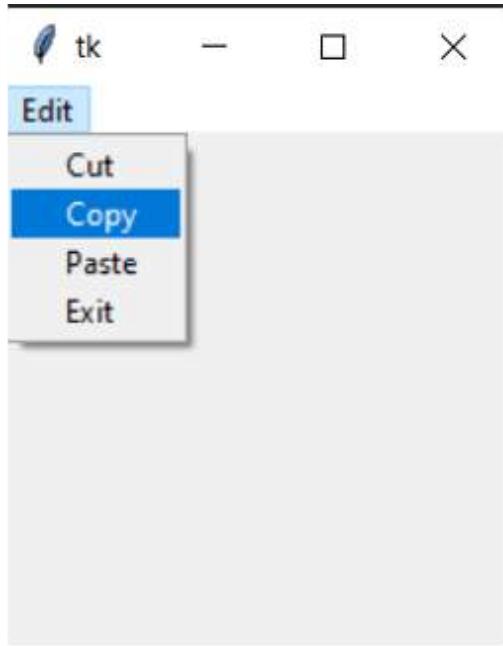


```
$ python mypythonguiprof.py
Paste
```

A terminal window showing the command '\$ python mypythonguiprof.py' followed by the word 'Paste' on a new line. The terminal has a dark background and light-colored text.

Figure 8.16: Output when Paste menu item is clicked; Paste is displayed on the console

Figure 8.17 shows the output when **Copy** menu item is clicked. As we can see, **Copy** is displayed to the console:



```
$ python mypythonguiprog.py
Copy
█
```

Figure 8.17: Output when Copy menu item is clicked; Copy is displayed on the console

**Note: The preceding code is covered in Program Name:
Chap8_Example5.py**

When the **Exit** menu item will be clicked, the GUI application will exit.

In the same example at position L2, we can specify the **activeborderwidth** of the widget to some value when it is under the mouse, as shown:

```
myedit_menu = Menu(mymenu_bar, tearoff=0,
activeborderwidth = 5) # L2
```

So, just observe the difference highlighted in blue color when the mouse is over the widget. Refer to [Figure 8.18](#):

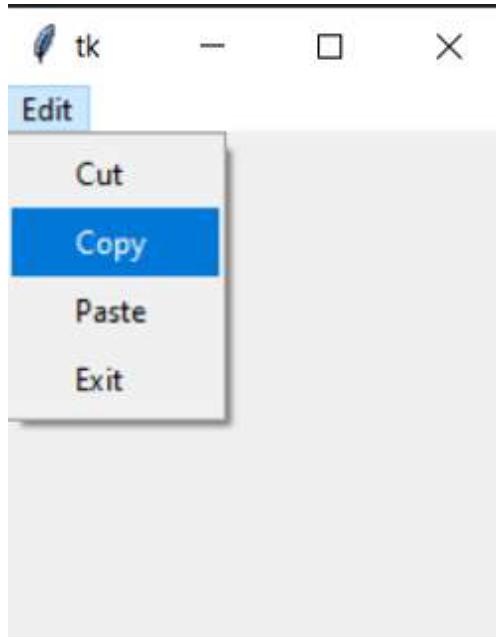


Figure 8.18: Output when activeborderwidth is set to value 5

[tkinter Menubutton widget](#)

This widget is a drop-down menu part that is shown to the user all the time. An option is provided to the user to select from different choices. It is associated with the menu which can display its choices when clicked by the user.

The syntax is as follows:

```
mymb1= Menubutton(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are **bg**, **bd**, **bitmap**, **activebackground**, **activeforeground**, **anchor**, **direction**, **cursor**, **disabledforeground**,

height, fg, highlightcolor, justify, image, menu, padx, pady, relief, text, state, textvariable, underline , wraplength, and width.

We have seen most of the options but some undiscussed options are as follows:

- **direction**: This option will specify the menu to be displayed in the specified direction to the button. It could be ABOVE, LEFT, or RIGHT where the menu will be displayed above, left, or right of the button.
- **highlightcolor**: This option will display the color when tkinter Menubutton widget is clicked.
- **image**: This option will set the image displayed on the tkinter Menubutton widget.
- **menu**: This option will display the menu associated with the menubutton.
- **text**: This option will display the text on the menubutton.
- **textvariable**: This option will allow controlling the above widget text at runtime by setting the control variable of string type at runtime to the text variable.

Refer to the following code:

```
from tkinter import *
myroot = Tk()
myroot.geometry('200x200')

# Create a menu button with options specified
gamelist = Menubutton(myroot, text='Games',
justify=CENTER, relief = 'groove')

# creating drop down menu which will become visible when
# the user will click the menu button
mygames = Menu(gamelist)
```

```
gamelist.config(menu=mygames)

# will add commands to the drop down menu
mygames.add_command(label='Cricket')
mygames.add_command(label='Football')
mygames.add_command(label='Badminton')
gamelist.pack()
myroot.mainloop()
```

Output when the Games button is clicked:

The output can be seen in *Figure 8.19*:

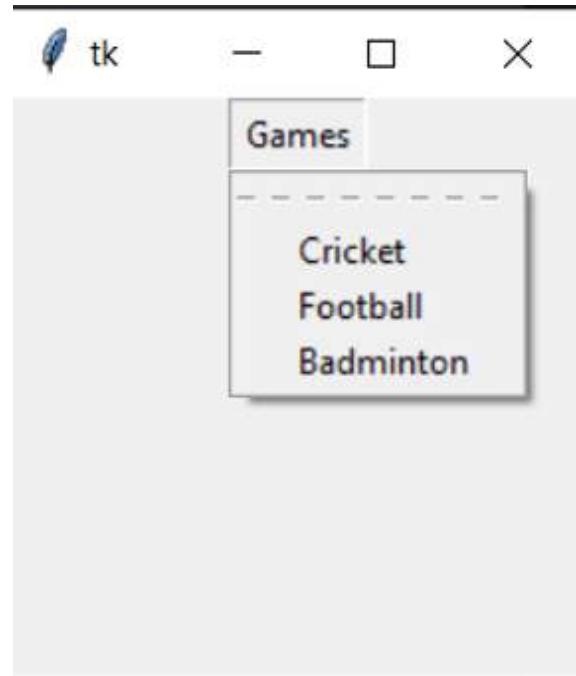


Figure 8.19: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example6.py**

We can add a checkbutton to the menubutton, as shown:

```
from tkinter import *
```

```
myroot = Tk()
myroot.geometry("300x150")

# creating menubutton
menubutton = Menubutton(myroot, text = "My Hobby", relief = FLAT)
menubutton.grid()

# creating pull down menu
menubutton.menu = Menu(menubutton, tearoff = 0)
menubutton["menu"] = menubutton.menu

# creating checkbutton
menubutton.menu.add_checkbutton(label = "Reading Books",
variable=IntVar())
menubutton.menu.add_checkbutton(label = "Playing Outdoor games",
variable = IntVar())
menubutton.pack()

myroot.mainloop()
```

Output when both hobbies are selected:

The output can be seen in *Figure 8.20*:

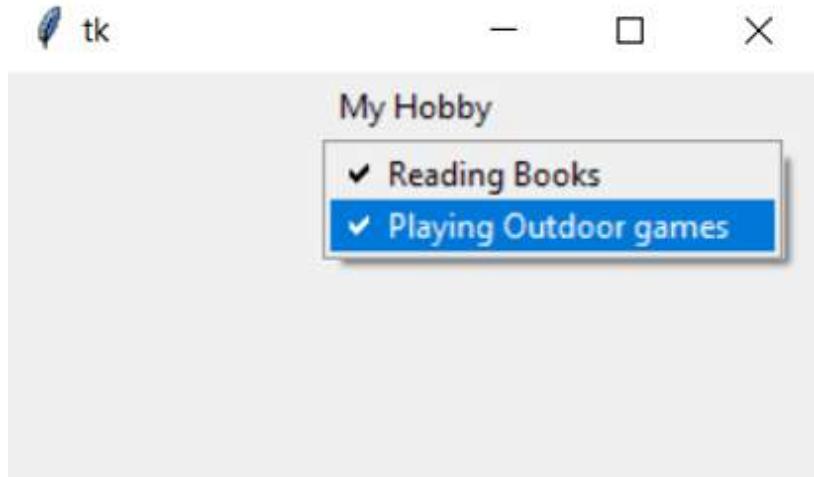


Figure 8.20: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example7.py**

Here, we can select both hobbies. However, we can force the user to select any one hobby by using radiobuttons, as shown:

```
from tkinter import *

myroot = Tk()
myroot.geometry("300x150")

# creating menubutton
menubutton = Menubutton(myroot, text = "My Hobby", relief = FLAT)
menubutton.grid()

# creating pull down menu
menubutton.menu = Menu(menubutton, tearoff = 0)
menubutton["menu"] = menubutton.menu

# creating radiobutton when clicked the color will be
# changed to green
myval1 = IntVar()

menubutton.menu.add_radiobutton(label = "Reading Books",
value = 1, variable= myval1, selectcolor = 'green')
menubutton.menu.add_radiobutton(label = "Playing Outdoor
games", value = 2, variable = myval1, selectcolor =
'green')

menubutton.pack()

myroot.mainloop()
```

Output:

The output can be seen in *Figure 8.21*:



Figure 8.21: Output

Note: The preceding code is covered in Program Name:
Chap8_Example8.py

tkinter Canvas widget

This widget will allow drawing structured graphics such as line, rectangle, circle, polygon, and so on, to the Python application. It can render graphs or plots and is a powerful widget for building GUI applications.

The syntax is as follows:

```
mycv1= Canvas(myroot, options...)
```

where,

- **myroot** is the parent window.
- Some of the lists of options that can be used as key-value pairs and are separated by commas are bg, bd, confine, cursor, height, relief, highlightcolor, scrollregion, width, xscrollincrement, yscrollincrement, xscrollcommand, and yscrollcommand.

We have seen most of the options but some undiscussed options are as follows:

- **confinne:** This method when set to true, will make the canvas unscrollable outside the scroll region.

- **scrollregion:** This method specified as tuple (N, S, E or W) represents the coordinates containing the canvas area. Here, N means top, S means bottom, E means right and W means left side.
- **xscrollincrement:** This option is used to set the scrolling dimension in horizontal direction. Setting some positive value, the scrolling limit will increase in multiple of the set value. If the value of this option is less than or equal to zero, then horizontal scrolling is unconstrained.
- **yscrollincrement:** This method is similar to **xscrollincrement** but the vertical movement is governed.
- **xscrollcommand:** This method should be the `.set()` method of the horizontal scrollbar if the canvas is scrollable.
- **yscrollcommand:** This method should be the `.set()` method of the vertical scrollbar if the canvas is scrollable.

The items supported by the above widget are line, rectangle, image, oval, and polygon. We shall see the usage of the tkinter Canvas widget to draw these items, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('400x400')
myroot.title('Linecreation')

# creation of canvas widget . draing a rectabgular area
myc1 = Canvas(myroot, width = 350, height = 350, bg =
'LightBlue') # L1
myc1.pack()

# drawing 2 lines
myline = myc1.create_line(0,0,300,150) # L2 (x1,y1,x2,y2)
mygreen_line = myc1.create_line(300,150,0,300, fill =
'Green') # L3(x1,y1,x2,y2)
```

```
myroot.mainloop() # display window until we press the  
close button
```

Output:

The output can be seen in *Figure 8.22*:

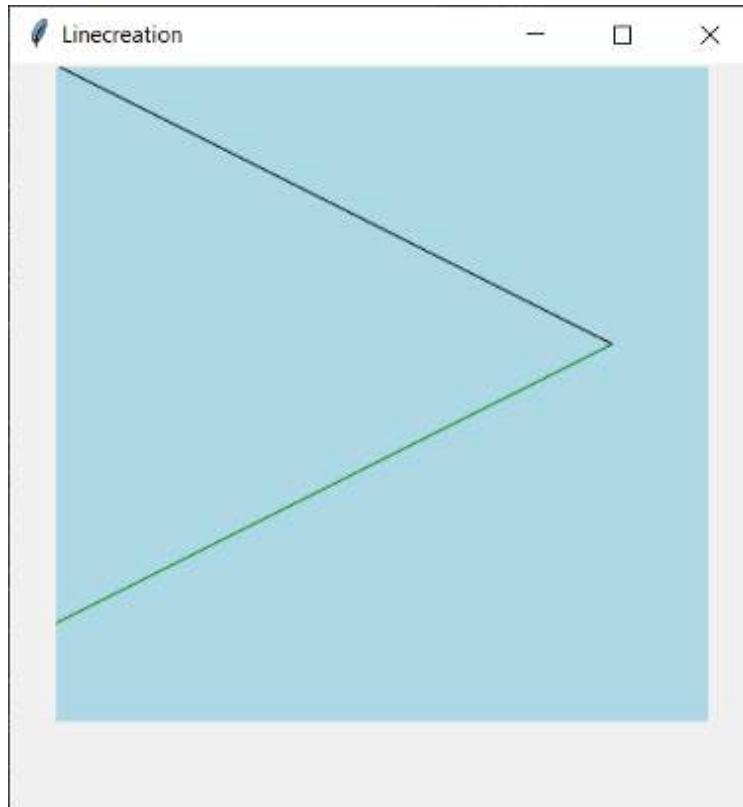


Figure 8.22: Output

**Note: The preceding code is covered in Program Name:
*Chap8_Example9.py***

In L1, we have created a canvas widget that contains the first parameter as the parent window, the 2nd, and 3rd parameters as width and height, and the last parameter as the background color. The widget can be customized by providing other widgets. 2 straight lines L2 and L3 are drawn using the `create_line()` method. The pixels are in the form of (x,y) pair and are referred from the top-left corner of the canvas and got the desired output as expected.

We can change the color of the line using the `itemconfigure()` method, as shown:

```
myc1.itemconfigure(myline, fill = 'Red') # changing the  
color of myline object
```

Refer to *Figure 8.23*:

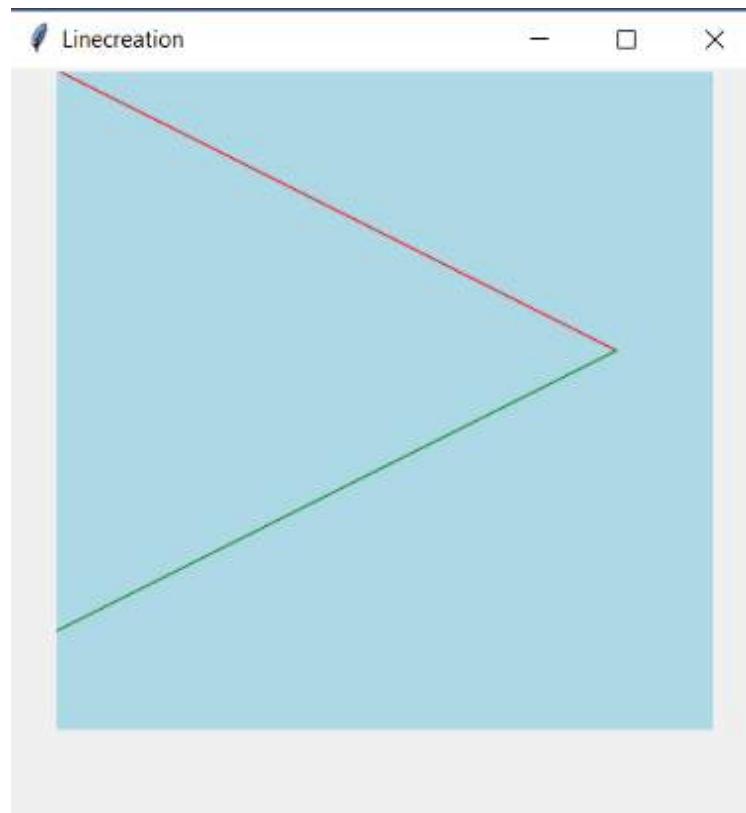


Figure 8.23: Output after changing the color of the line from topleft as Red

We can get the coordinates associated with the line object by using **coords()** method, as shown:

```
print(myc1.coords(myline))
```

Refer to *Figure 8.24*:

```
$ python mypythonguiprof.py  
[0.0, 0.0, 300.0, 150.0]
```

Figure 8.24: Output displaying the coordinates associated with the line object

We can also change the coordinates of the line using the **coords()** method. Just add the following line and comment on the previous command, as shown:

```
# print(myc1.coords(myline))
myc1.coords(myline, 0,0,175,175,350,0)
```

Refer to *Figure 8.25*:

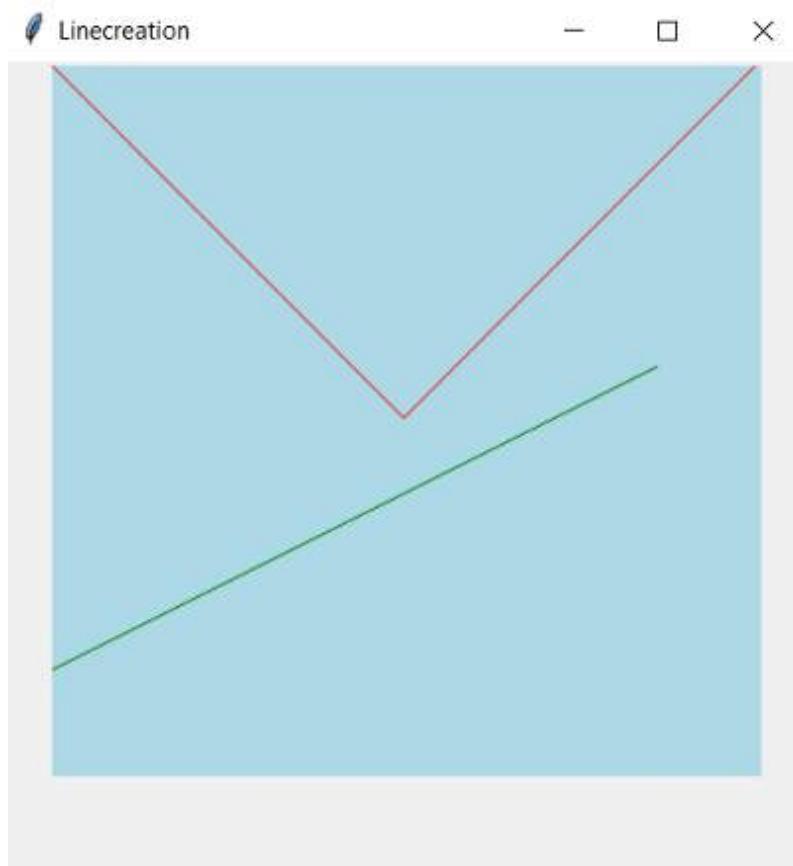


Figure 8.25: Output displaying the change of coordinates using cords method

Here, we have created 3 pairs of coordinates.

We can also smoothen the line. Just add the following line of code, as shown:

```
myc1.itemconfigure(myline, smooth = True)
```

Refer to *Figure 8.26*:

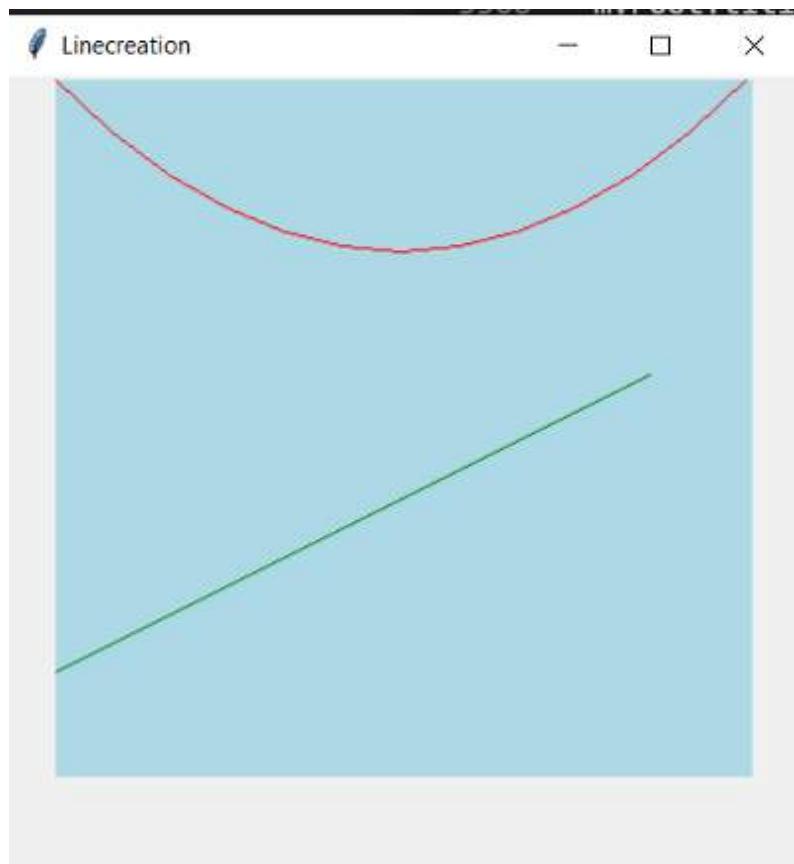


Figure 8.26: Output displaying line smoothen

Here, the canvas will draw multiple line segments, in order to create a smooth appearance of the line.

We can control how many line segments it uses to try and represent this line by configuring the **splinesteps** option. So, add the following line in the code:
`myc1.itemconfigure(myline, splinesteps = 6)`

Refer to [*Figure 8.27*](#):



Figure 8.27: Output with configuration of splinesteps options

So, this will draw the line by using 6 steps to try and smooth it up.

We can delete the line object by adding the following line of code:
`myc1.delete(myline)`

Refer to [Figure 8.28](#):

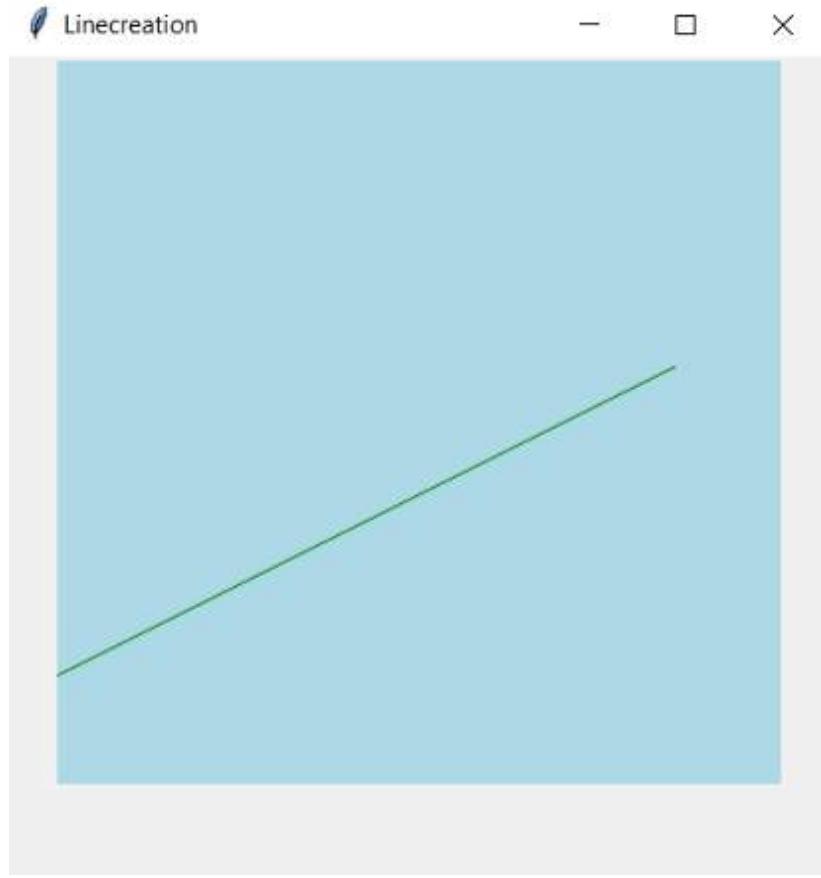


Figure 8.28: Output after deleting the line object

Now, we shall see how to create an arc using a canvas widget:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('300x300')
myroot.title('arccreation')

# canvas widget creation
myc1 = Canvas(myroot, width = 300, height = 300, bg =
'LightBlue')
myc1.pack()

# arc creations
```

```
myc1.create_arc(50,50,150,150)
myc1.create_arc(120,120,200,200, extent = 120)

#fill: arc interior filled with Red color
myc1.create_arc(180,180,250,250, extent = 120, style =
CHORD, fill = 'Red')

# start angle par: start location in degrees whose default
is 0 deg

# extent angle par: from the start location the no. of
degrees to extend the arc whose default is 90 deg

# style par: arc style to draw could be pieslice(default),
chord and arc

myc1.create_arc(180,250,270,270, start = 50, extent = 120,
style = ARC)

myroot.mainloop() # display window until we press the
close button
```

Output:

Refer to *Figure 8.29*:

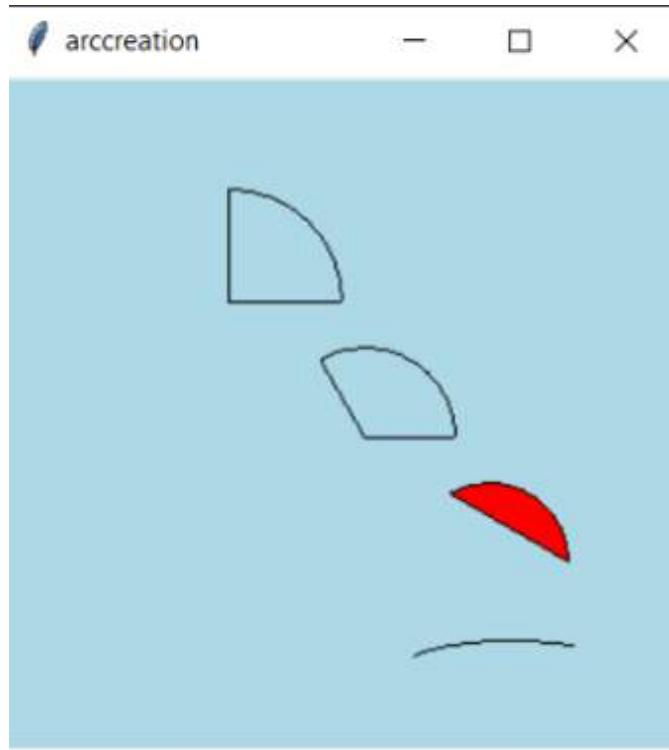


Figure 8.29: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example10.py**

We can create an image item using Canvas, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('360x360')
myroot.title('ImageCanvas')

myc1 = Canvas(myroot, width = 360, height = 360, bg =
'LightBlue')
myc1.pack()

myphoto = PhotoImage(file = 'butterfly1.gif')
myc1.create_image(0,0,image = myphoto, anchor = NW)
```

```
myroot.mainloop() # display window until we press the  
close button
```

Output:

Refer to *Figure 8.30*:

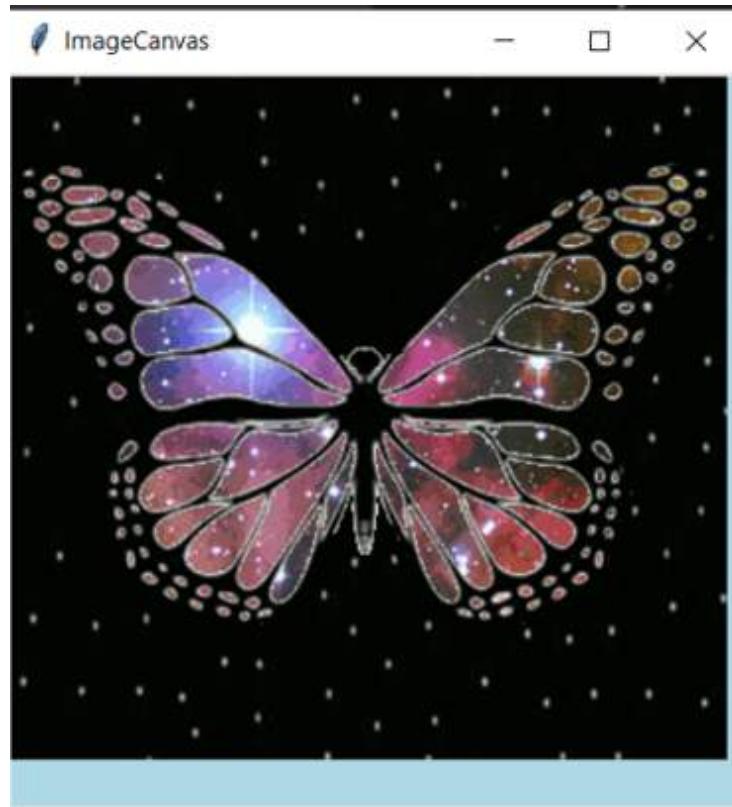


Figure 8.30: Output of Chap8_Example11.py

**Note: The preceding code is covered in Program Name:
Chap8_Example11.py**

In the above code, we have created a canvas with the first parameter as the parent window and other options such as height, width, and backgroundcolor. The photoimage available in the tkinter package will help us to keep our image as an item over it. The image name location is referred to in Photoimage and is placed over the canvas.

We can create a rectangle using Canvas, as shown:

```
from tkinter import * # importing module
```

```
myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('300x300')
myroot.title('rectanglecreation')

#canvas creation
myc1 = Canvas(myroot, width = 300, height = 300, bg =
'LightBlue')
myc1.pack()

# rectangle creation
myrect = myc1.create_rectangle(100,100,300,300, fill =
'Red', outline = 'Blue')

myroot.mainloop() # display window until we press the
close button
```

Output:

Refer to *Figure 8.31*:

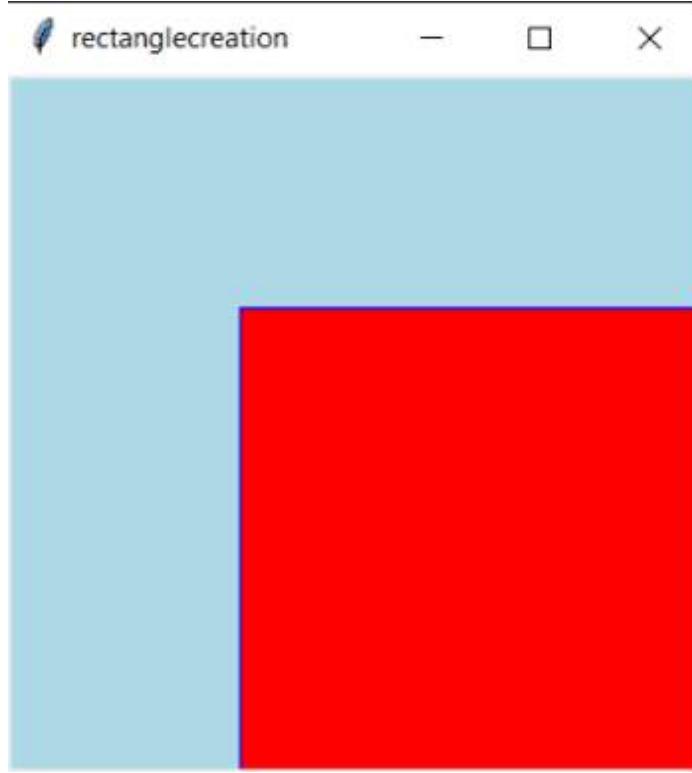


Figure 8.31: Output of Chap8_Example12.py

**Note: The preceding code is covered in Program Name:
Chap8_Example12.py**

In the above code, initially, we have created a canvas with the first parameter as the parent window and other options as height, width, and backgroundcolor. Then we created a rectangle with 2 pairs of coordinates, mentioned along with other options such as fill and outline.

We can also create an ellipse or circle at the given coordinates. The 2 pairs of coordinates are taken with the top left and bottom right corners of the bounding rectangle for the oval, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('350x350')
myroot.title('circlecreation')
```

```
# canvas object creation
myc1 = Canvas(myroot, width = 350, height = 350, bg =
'LightBlue')
myc1.pack()

# rectangle object creation
myrect = myc1.create_rectangle(100,100,350,350, fill =
'Red', outline = 'Red')

#oval object creation
myc1.create_oval(100,100,350,350, fill = 'Yellow', outline
= 'Yellow')

myroot.mainloop() # display window until we press the
close button
```

Output:

Refer to *Figure 8.32*:

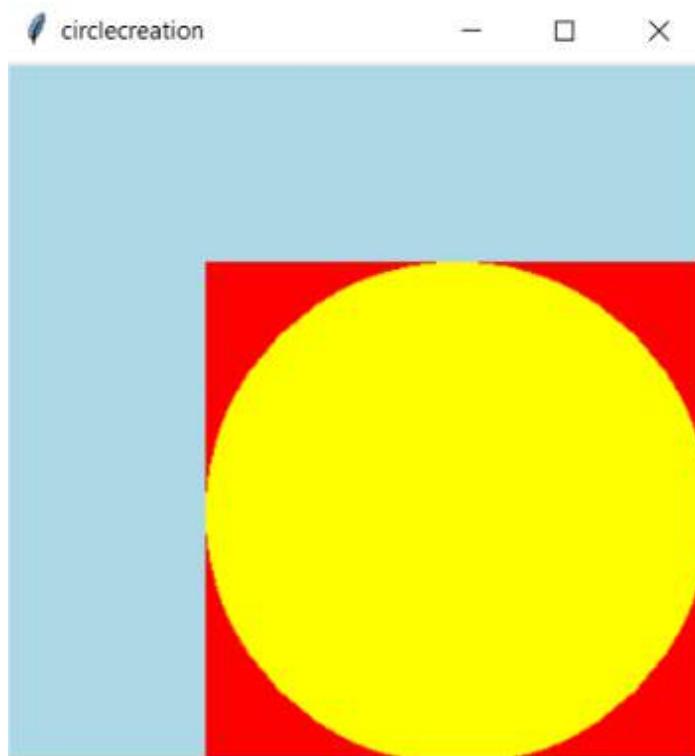


Figure 8.32: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example13.py**

We can create a polygon but it should have at least 3 vertices, as shown:

```
from tkinter import *
class CreatePolygon(Frame):
    def __init__(self, myroot=None):
        # myroot object is initialised
        super().__init__(myroot) # Calling
Frame.__init__(myroot)
        self.myroot = myroot # Update the myroot object
after Frame() makes necessary changes to it

    def mycreateCanvas(self, mycanvas_width,
mycanvas_height):
        # Creating canvas object
        mycanvas = Canvas(self.myroot, bg="LightBlue",
width=mycanvas_width, height=mycanvas_height)
        return mycanvas

    def mycreate_polygon(self, mycanvas):
        mypoints = [100,200,200,100,250,350,100,200]
        mycanvas.create_polygon(mypoints,fill = 'Yellow',
outline = 'Red', width = 2) # polygon creation
        return mycanvas

# Create our myroot object to the Application
myroot = Tk()
myroot.title('polygoncreation')
```

```
# creating create_polygon object
myobj = CreatePolygon(myroot=myroot)
mycanvas = myobj.mycreateCanvas(400, 400)
mycanvas = myobj.mycreate_polygon(mycanvas)

# The items are packed into the canvas
mycanvas.pack()
# Start the mainloop
myobj.mainloop()
```

Output:

Refer to *Figure 8.33*:

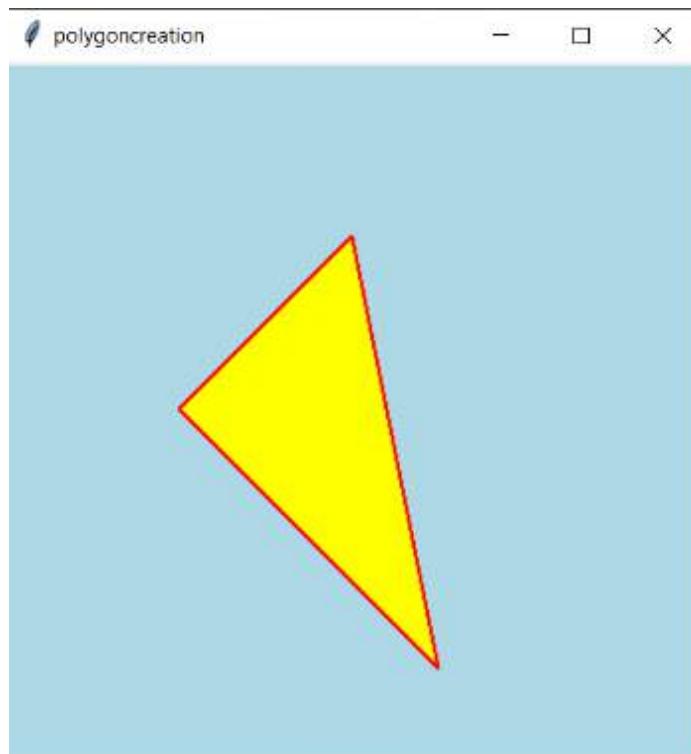


Figure 8.33: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example14.py**

We can also write text in Canvas, as shown:

```
from tkinter import * # importing module

myroot = Tk() # window creation and initialize the
interpreter
myroot.geometry('300x200')
myroot.title('Textwriting')

# canvas object creation
myc1 = Canvas(myroot, width = 300, height = 200, bg =
'LightBlue')

# Create text
myc1.create_text(10, 100, anchor=W, font="Helvetica",
text="Hey! I am writing text in Canvas")
myc1.pack()

myroot.mainloop() # display window until we press the
close button
```

Output:

Refer to *Figure 8.34*:

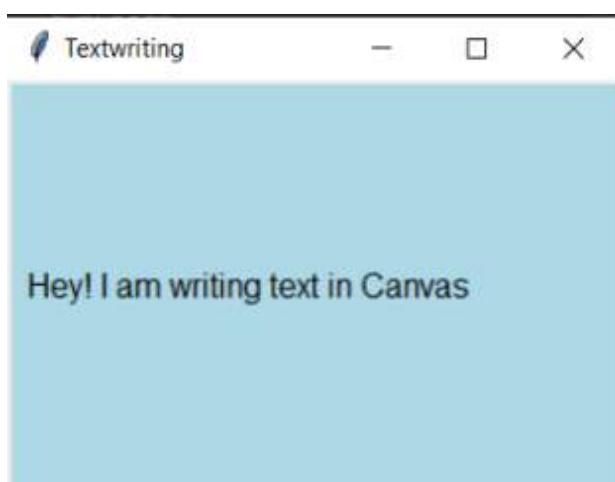


Figure 8.34: Output

**Note: The preceding code is covered in Program Name:
Chap8_Example15.py**

Conclusion

In this chapter, we learned about creation of pop-up, top-level, and pull-down menus with the help of the tkinter Menu widget. We also saw example of adding radiobutton and checkbutton to the menu. The tearoff option, `entryconfig()` method, adding a specific type of item using the key itemType or a positional argument was well explored. We also viewed a widget which was associated with the menu and was provided to the user to select, from different choices, viz. tkinter Menubutton widget. We also saw example of adding radio and checkboxes to this menubutton widget. Finally, we explored the concepts of drawing different graphics such as lines, rectangles, arc, polygon, text and create of image item1 with the help of the tkinter Canvas widget.

Points to remember

- Menus can be created using tkinter menu widget. A hierarchical menu structure is created by nesting of menus. Users are provided with a way to access additional commands or options using menus.
- Button that displays a drop-down menu is created by using tkinter menubutton widget. It is associated with the menu which can display its choices when clicked by the user. Users are provided with a way to access additional commands or options using menubutton widget.
- tkinter canvas widget will allow drawing structured graphics such as line, rectangle, circle, polygon, and so on, to the Python application. We need to use the Canvas method on the parent widget to create a canvas, and then add elements to it with the **create_ methods** such as `create_line`, `create_rectangle`, `create_text` and `create_image`.
- Meaningful titles can be given to menus and menubuttons. Users will find it simpler to understand what each menu item or menu button accomplishes as a result.

- In menus, it is important to group relevant commands together which will make the job of user simpler for locating the commands.
- For menus and menubuttons, use icons.
- Maintain a consistent look for your menus and menubuttons. The GUI will look sleeker and user-friendly as a result.
- Thoroughly test your menus and menubuttons. Verify that they perform as planned and do not introduce any issues.

Questions

1. Explain the tkinter Menu Widget in detail.
2. Which widget is a top-level menu and provides options such as File, Edit, quit, in the application?
3. Explain the tkinter Menu Widget in detail and its syntax with a suitable program to justify your answer.
4. Write a program to create a menubar and then create the items Welcome! and Quit!
5. Explain the tkinter Menubutton Widget in detail.
6. Which widget is a drop-down menu part that is shown to the user all the time? Explain in detail.
7. Which widget has an option that allows the user to select from different choices?
8. Explain the tkinter Canvas Widget with its syntax and a short program to justify your answer.
9. Which widget allows to plot structured graphics like line, rectangle, circle, and polygons? Explain the widget in detail.
10. How to render graphs or plots for building GUI applications? Explain in detail.

CHAPTER 9

Handling File Selection in tkinter

Introduction

In GUI applications, handling file selection in tkinter is frequently required since we must let the user choose a file for processing or display. For instance, we can allow the user to choose an image file to display in an image viewer application, or choose a text file to load and edit in a text editor application.

To assist with managing file selection dialogs, tkinter offers the `filedialog` module. This can make it much simpler to implement this feature in our application. The selected file or directory's path is returned by the `filedialog` module's methods for selecting, saving, and selecting files and directories. We may make sure that our application is cross-platform compatible and that the user can select files using a typical file selection dialogue by utilizing the `filedialog` module. This can improve the usability and appearance of application.

Structure

In this chapter, we will discuss the following topics:

- Handling file selection in tkinter
 - **askdirectory**
 - **askopenfile**
 - **askopenfilename**

- **askopenfilenames**
- **asksaveasfile**
- **asksaveasfilename**

Objectives

After reading this chapter, the reader will learn about handling file selection with different dialogs for opening a file, saving a file, and so on, with help of multiple examples created with various Python applications.

Handling file selection in tkinter

We come across the requirements of opening a file, saving a file, and opening a directory sort of operations. In such cases, the tkinter provides a dialog containing a small file browser called a file dialog, which is a part of the **filedialog** module. There are a different set of functions which creates file dialogs. The common arguments taken by each function are:

- **title**: This parameter will specify the title of the dialog window.
- **parent**: This parameter will specify the optional parent widget.
- **initialdir**: This parameter will specify the directory in which the file browser should start.
- **filetypes**: This parameter will specify the list of tuples each with a label and matching pattern for filtering the visible files supported by the application.

Some functions take additional options such as:

- **initialfile**: It is a default file path to select.
- **defaultextension**: It is a file extension string that is appended automatically to the filename if the user does not do it.

Some functions return a file object taking mode argument, specifying the file open mode.

Now, we shall discuss each function one by one.

askdirectory: This function will return the directory path as a string and it displays only directories.

```
from tkinter import *
from tkinter.filedialog import askdirectory
myroot = Tk()
myroot.title('Askdirectory')
myroot.geometry('300x100')
def mydisplay():
    myroot.directory = askdirectory()

mybtn1 = Button(myroot, text = 'MyDirectoryOpen', command
= mydisplay)
mybtn1.pack(pady = 10)
myroot.mainloop()
```

Output:

Figure 9.1 shows the default output:

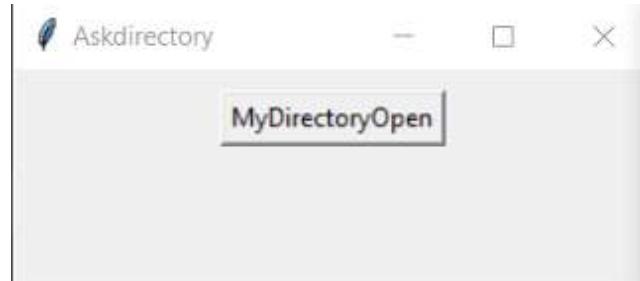


Figure 9.1: Default output

Figure 9.2 shows the output when the **MyDirectoryOpen** button is clicked:

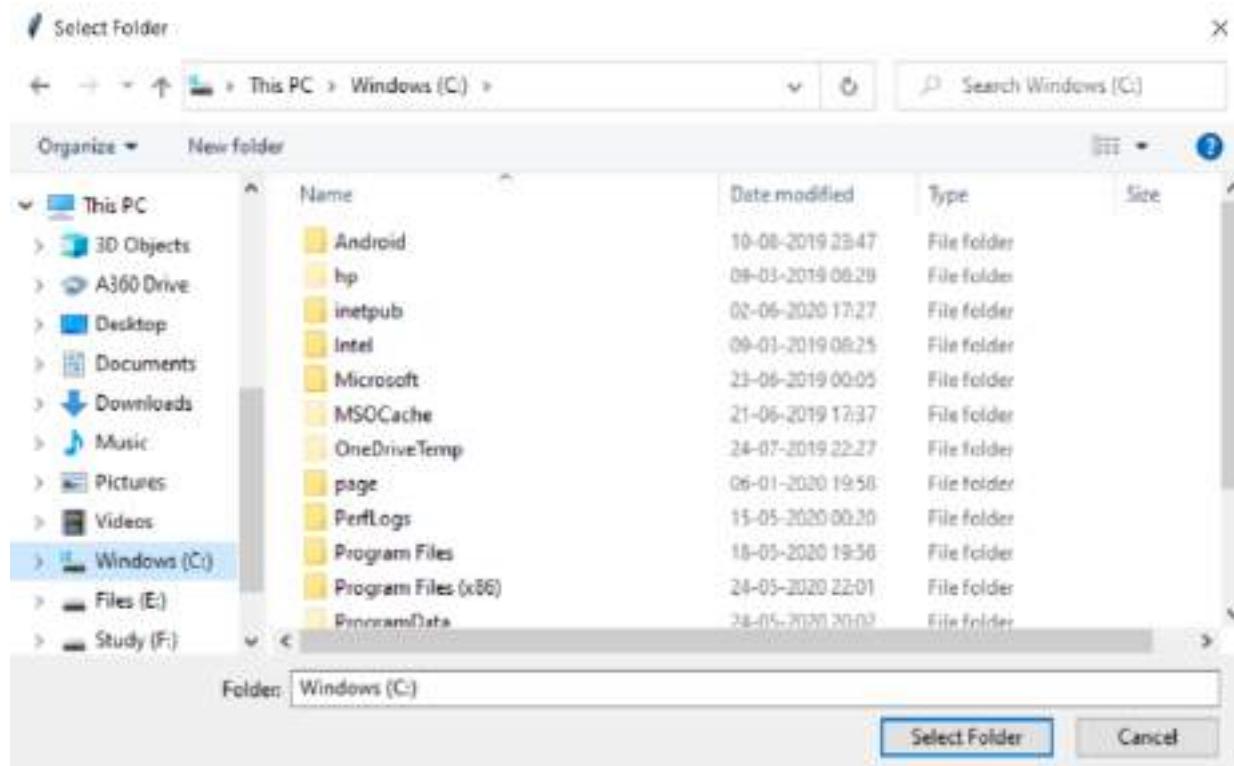


Figure 9.2: Output when the MyDirectoryOpen button is clicked

**Note: The preceding code is covered in Program Name:
Chap9_Example1.py**

askopenfile: This function will return a file handle object and it allows only the selection of existing files.

```
from tkinter import *
from tkinter.filedialog import askopenfile
myroot = Tk()
myroot.title('AskOpenFile')
myroot.geometry('300x100')
def myopen_file():
    myfile = askopenfile(mode ='r', filetypes =[('All Python Files', '*.py')])
    if myfile is not None:
        content = myfile.read() # read all the file
        contents
```

```
print(myfile.name) # display the file name

mybtn1 = Button(myroot, text = 'MyAskOpenFile', command =
myopen_file)
mybtn1.pack(pady = 10)

myroot.mainloop()
```

Output:

Figure 9.3 shows the default output:

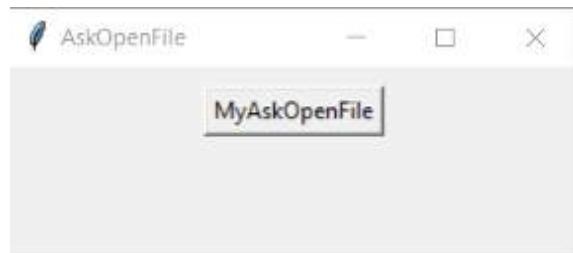


Figure 9.3: Default output

Figure 9.4 shows the output when the **MyAskOpenFile** button is clicked:

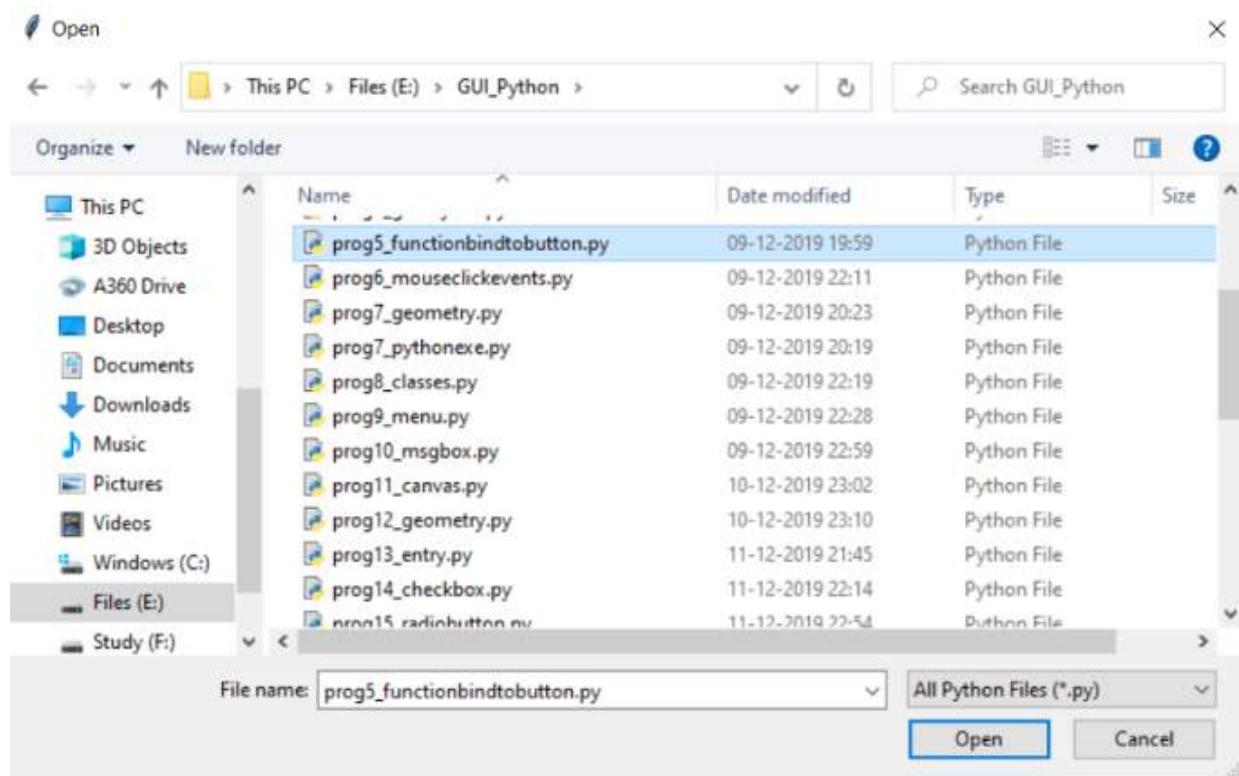


Figure 9.4: Output when the MyAskOpenFile button is clicked

**Note: The preceding code is covered in Program Name:
Chap9_Example2.py**

Output when any .py file is chosen and displays that file name:

E:/GUI_Python/prog5_functionbindtobutton.py

askopenfilename: This function will return the file path as a string and it allows only the selection of existing files.

```
from tkinter import *
from tkinter.filedialog import askopenfilename
myroot = Tk()
myroot.title('AskOpenFileName')
myroot.geometry('300x100')
def myopen_file():
    myfile = askopenfilename()
    print(myfile) # display the file name
```

```
mybtn1 = Button(myroot, text = 'MyAskOpenFileName',  
command = myopen_file)  
mybtn1.pack(pady = 10)  
myroot.mainloop()
```

Output:

Figure 9.5 shows the default output:

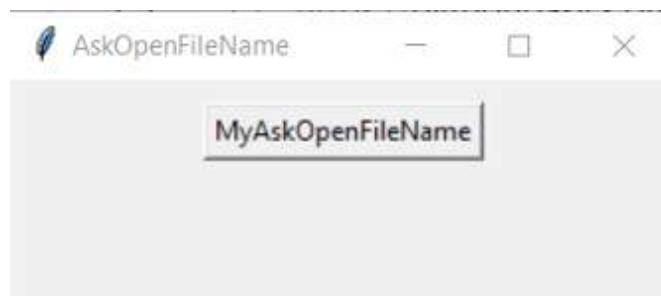


Figure 9.5: Default output

Figure 9.6 shows the output when the MyAskOpenFileName button is clicked:

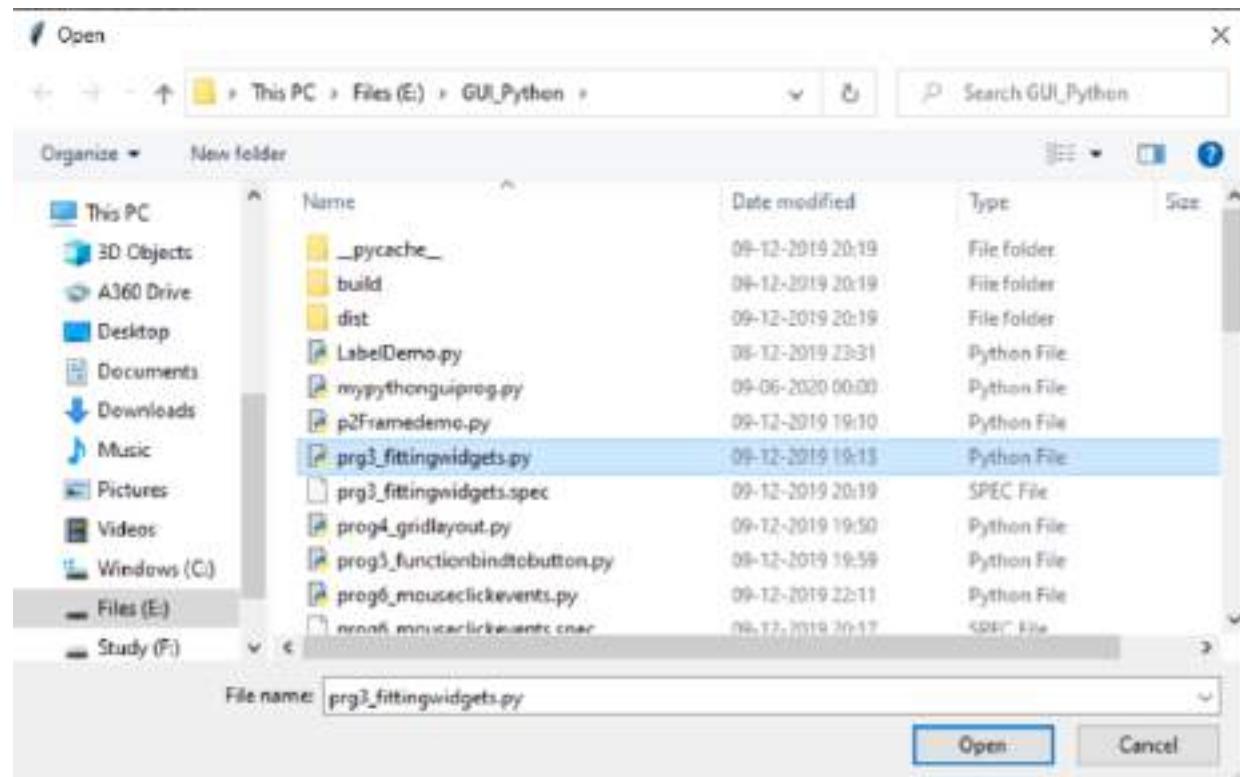


Figure 9.6: Output when the MyAskOpenFileName button is clicked

**Note: The preceding code is covered in Program Name:
Chap9_Example3.py**

Output when any .py file is chosen and displays that file name:

E:/GUI_Python/prg3_fittingwidgets.py

In the above code, we are creating an open file dialog that asks for a filename and returns the selected dialog name.

askopenfilenames: This function will return file paths as the list of strings and it allows multiple selections.

```
from tkinter import *
from tkinter.filedialog import askopenfilenames
myroot = Tk()
myroot.title('AskOpenFileNames')
myroot.geometry('300x100')
def myopen_files():
    myfile_list=[]
    myfiles =
    askopenfilenames(initialdir="E:\\GUI_Python",
title="Select Python files")
    for file in myfiles:
        myfile_list.append(file)
    print(myfile_list)

mybtn1 = Button(myroot, text = 'MyAskOpenFileNames',
command = myopen_files)
mybtn1.pack(pady = 10)

myroot.mainloop()
```

Output:

Figure 9.7 shows the default output:

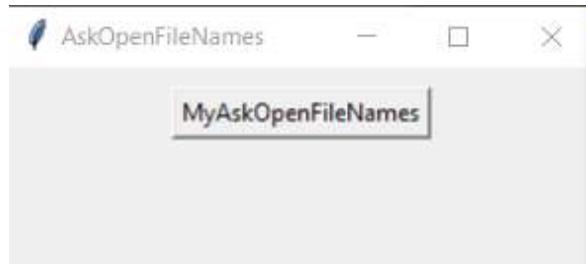


Figure 9.7: Default output of Chap9_Example4.py

Figure 9.8 shows the output when the **MyAskOpenFileName** button is clicked:

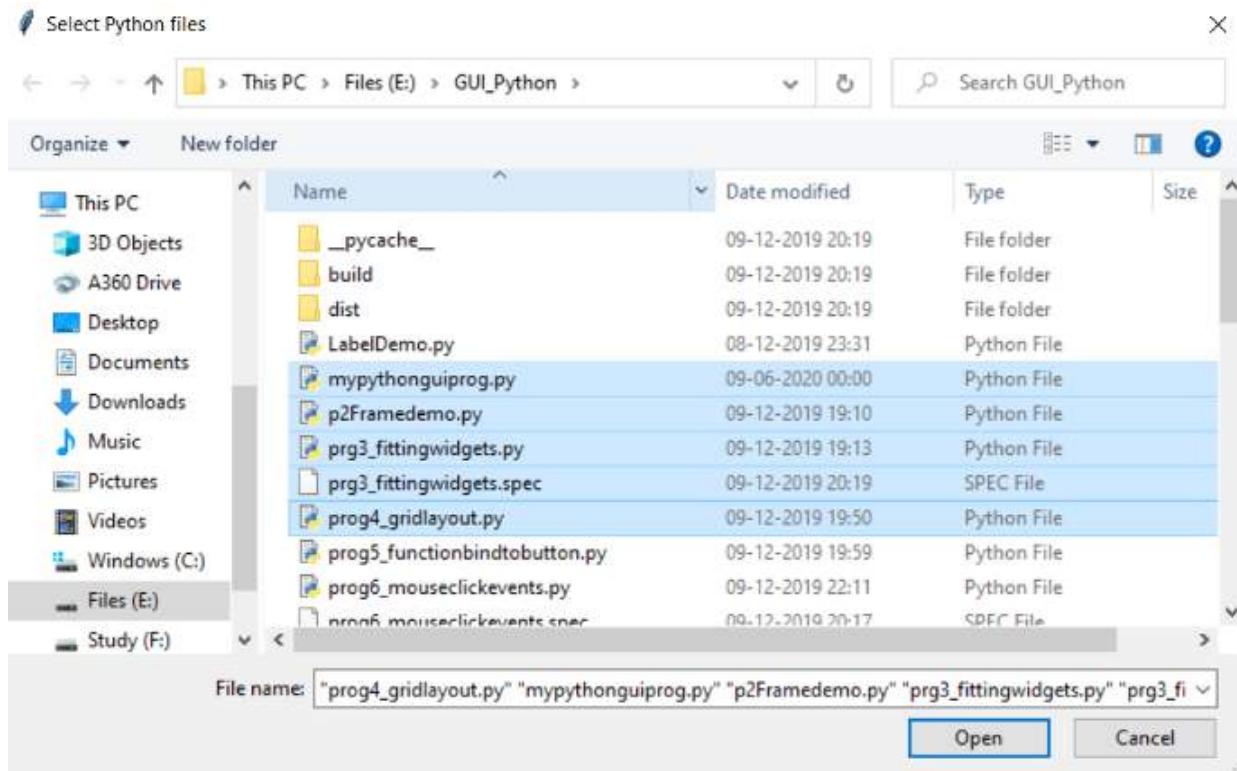


Figure 9.8: Output when the MyAskOpenFileName button is clicked

**Note: The preceding code is covered in Program Name:
Chap9_Example4.py**

Output when any .py file is chosen and displays that file name.

[‘E:/GUI_Python/mypythonguiprogram.py’, ‘E:/GUI_Python/p2Framedemo.py’,
‘E:/GUI_Python/prg3_fittingwidgets.py’,

```
'E:/GUI_Python/prg3_fittingwidgets.spec',
'E:/GUI_Python/prog4_gridlayout.py']
```

In the above code, we are creating an open file dialog that asks for filenames and returns the selected dialog names.

asksaveasfile: This function will return a file handle object and it allows new file creations and confirmation on existing files to be prompted.

```
from tkinter import *
from tkinter.filedialog import asksaveasfile
myroot = Tk()
myroot.title('AskSaveasFile')
myroot.geometry('350x150')
def saveas():
    mytxt = mye1.get()
    myfiles = [ ('All Files', '*.*'),
                ('Python Files', '*.py'),
                ('Text Document', '*.txt')]
    myfile1 = asksaveasfile(filetypes = myfiles,
                           defaultextension = myfiles)
    myfile1.write(mytxt)

mye1 = Entry(myroot, font=('Verdana',12))
mye1.place(x=10,y=10,width=200,height=100)

mybtn1 = Button(myroot, text='SaveasFile', font=
('Courier',10), width=10, bd=10, command =saveas)
mybtn1.place(x=220,y=110)

myroot.mainloop()
```

Output:

Figure 9.9 shows the default output:

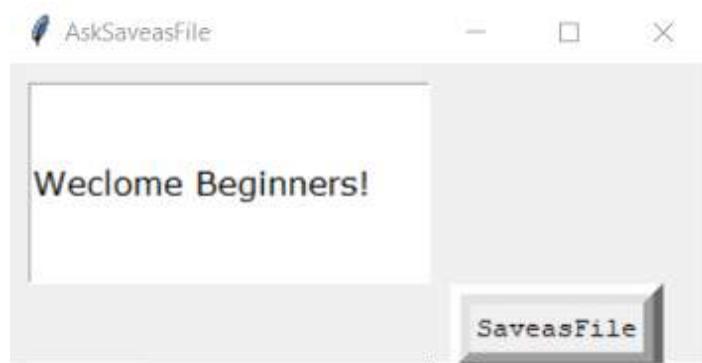


Figure 9.9: Default output

Figure 9.10 shows the output when the **SaveasFile** button is clicked:

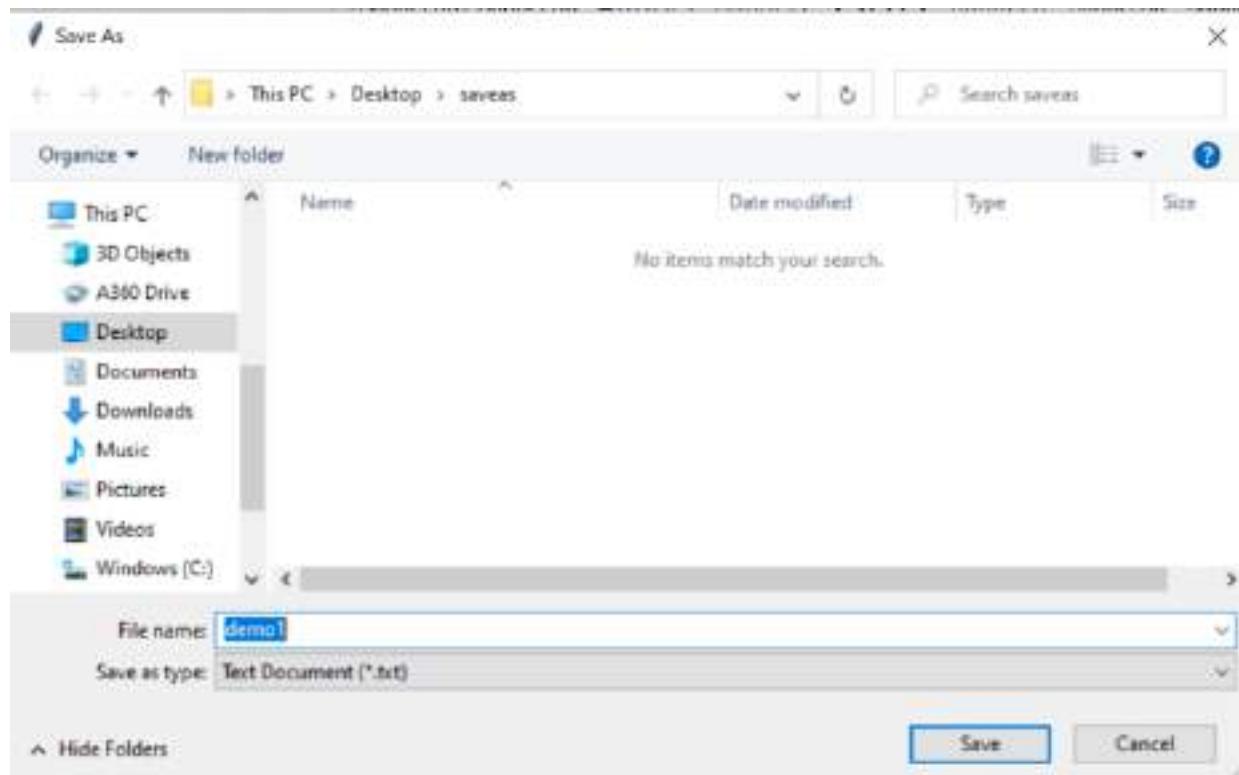


Figure 9.10: Output when the **SaveasFile** button is clicked

We can see that there are no **.txt** files present in the above folder. When the **Save** button is clicked, the file is saved as **demo1.txt** in the above folder, as shown in **Figure 9.11**:

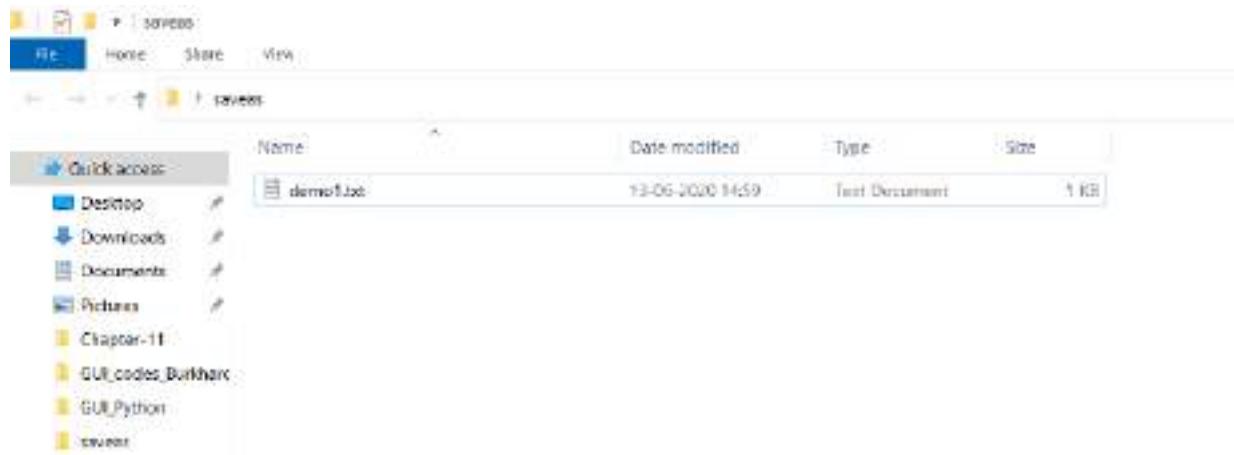


Figure 9.11: Output after saving as demo1.txt file

The contents inside the folder is shown in [Figure 9.12](#):

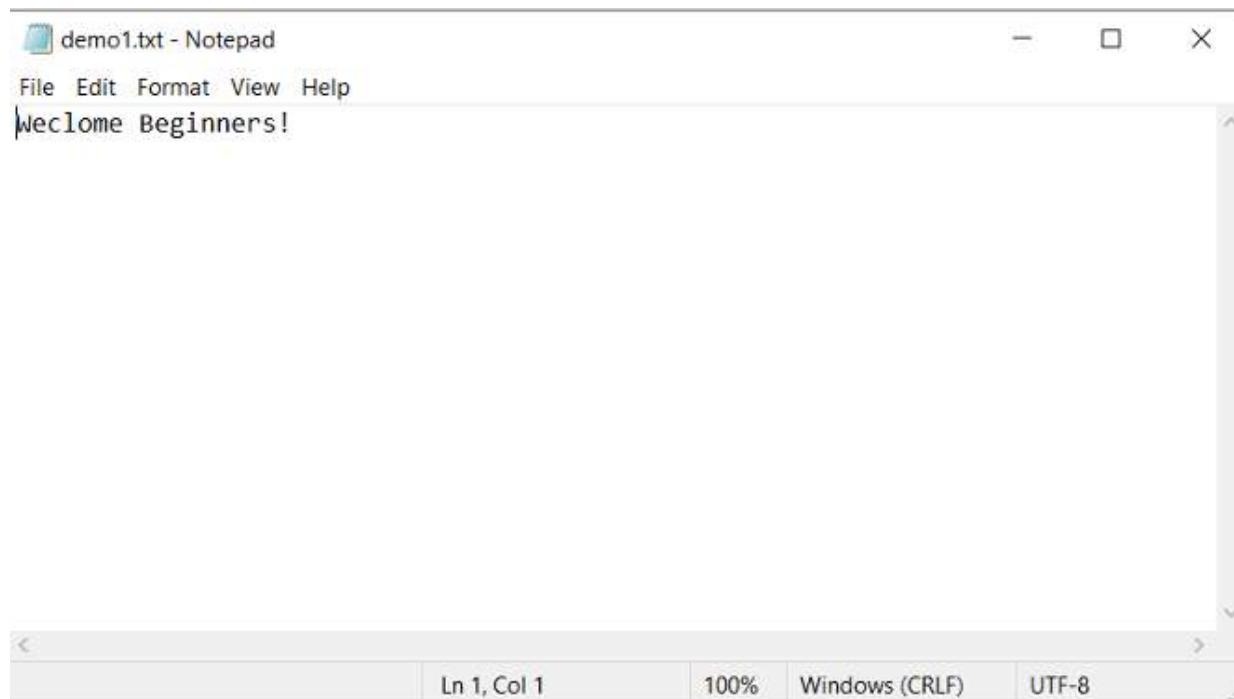


Figure 9.12: Output displaying the contents inside demo1.txt file

**Note: The preceding code is covered in Program Name:
Chap9_Example5.py**

asksaveasfilename: This function will return the file path as a string and it allows new file creations and confirmation on existing files to be prompted.

```
from tkinter import *
```

```

from tkinter import messagebox
from tkinter.filedialog import asksaveasfilename
myroot = Tk()
myroot.title('AskSaveasFileName')
myroot.geometry('350x150')
def saveas():
    mytxt = mye1.get()
    myfiles = [ ('All Files', '*.*'),
                ('Python Files', '*.py'),
                ('Text Document', '*.txt')]
    myfile1 = asksaveasfilename(filetypes = myfiles,
                                defaultextension = myfiles, confirmoverwrite = False)
    fname = myfile1
    if fname != '':
        try:
            myfile1 = open(fname, "a+")
            myfile1.write('\n' +str(mytxt))
            myfile1.close()
            messagebox.showinfo('Data Writing!', 'Data has
been appended')
        except:
            print('There is no such file')

mye1 = Entry(myroot, font=('Verdana',12))
mye1.place(x=10,y=10,width=200,height=100)
mybtn1 = Button(myroot, text='SaveasFileName', font=
('Courier',10), width=14, bd=10, command =saveas)
mybtn1.place(x=210,y=110)

myroot.mainloop()

```

Output:

Figure 9.13 shows the default output:



Figure 9.13: Default output

Figure 9.14 shows the output when the **SaveasFileName** button is clicked:

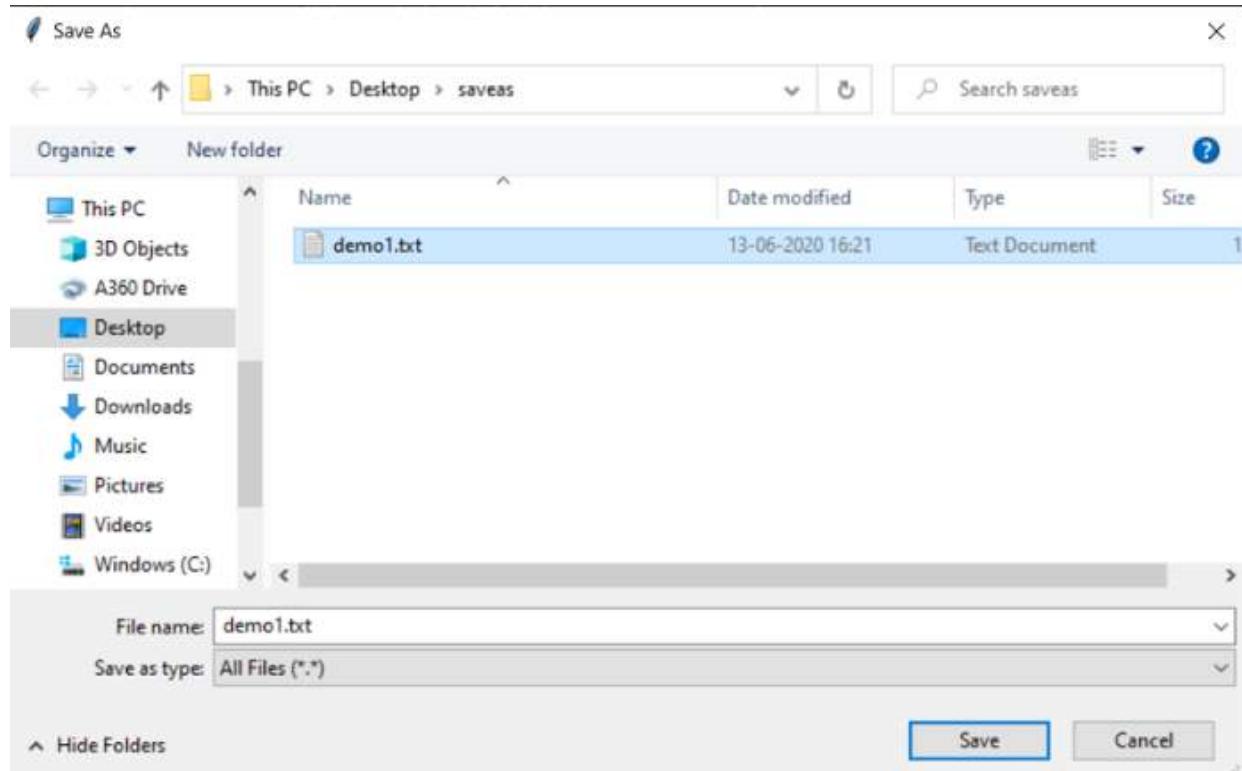


Figure 9.14: Output when the SaveasFileName button is clicked

On clicking the **Save** button, the following message shown in *Figure 9.15* will pop up:

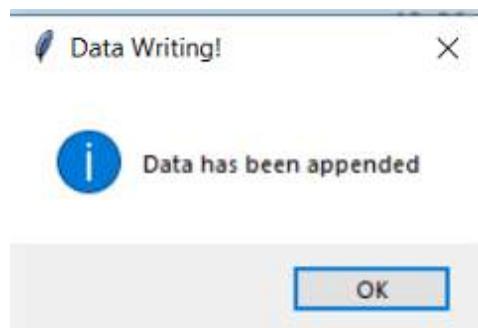


Figure 9.15: Output display after clicking the Save button

Here, we are appending the already created text file. So, the data inside the file **demo1.txt** file is shown in [Figure 9.16](#):

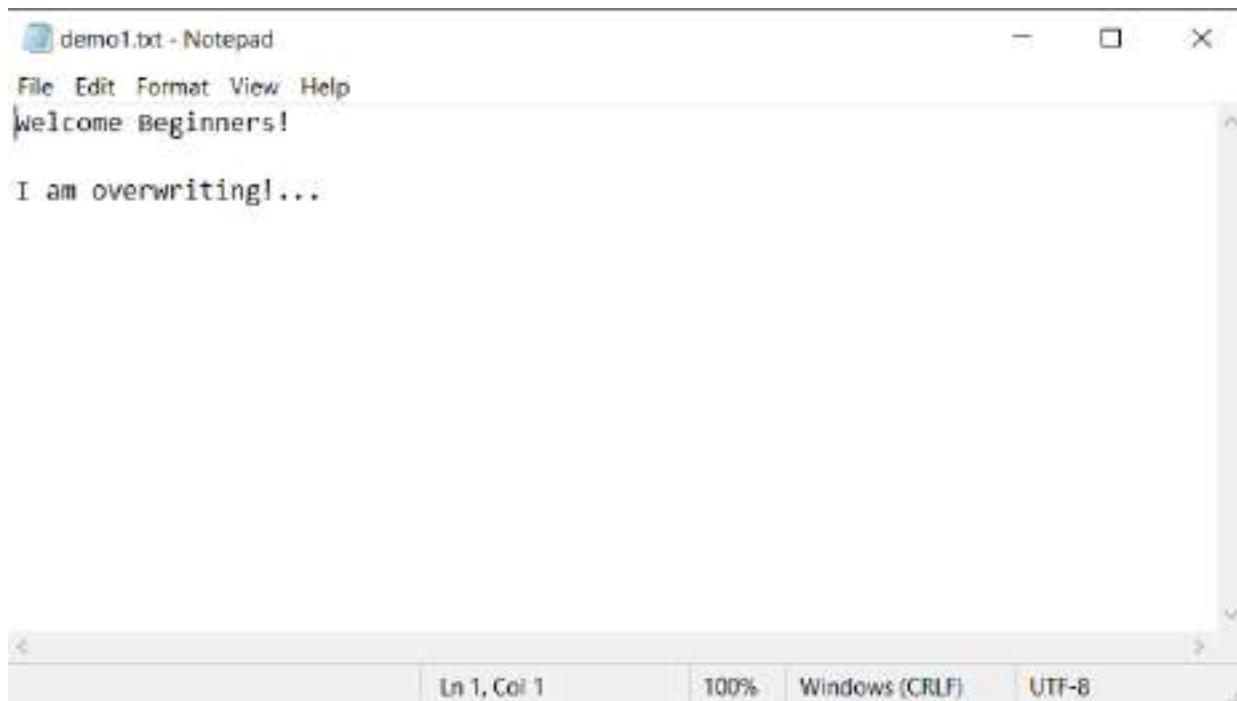


Figure 9.16: Contents display inside demo1.txt file appending the newly added text

If the file is not created, the new text file, say '**demo2.txt**', will be created, as shown in [Figure 9.17](#):

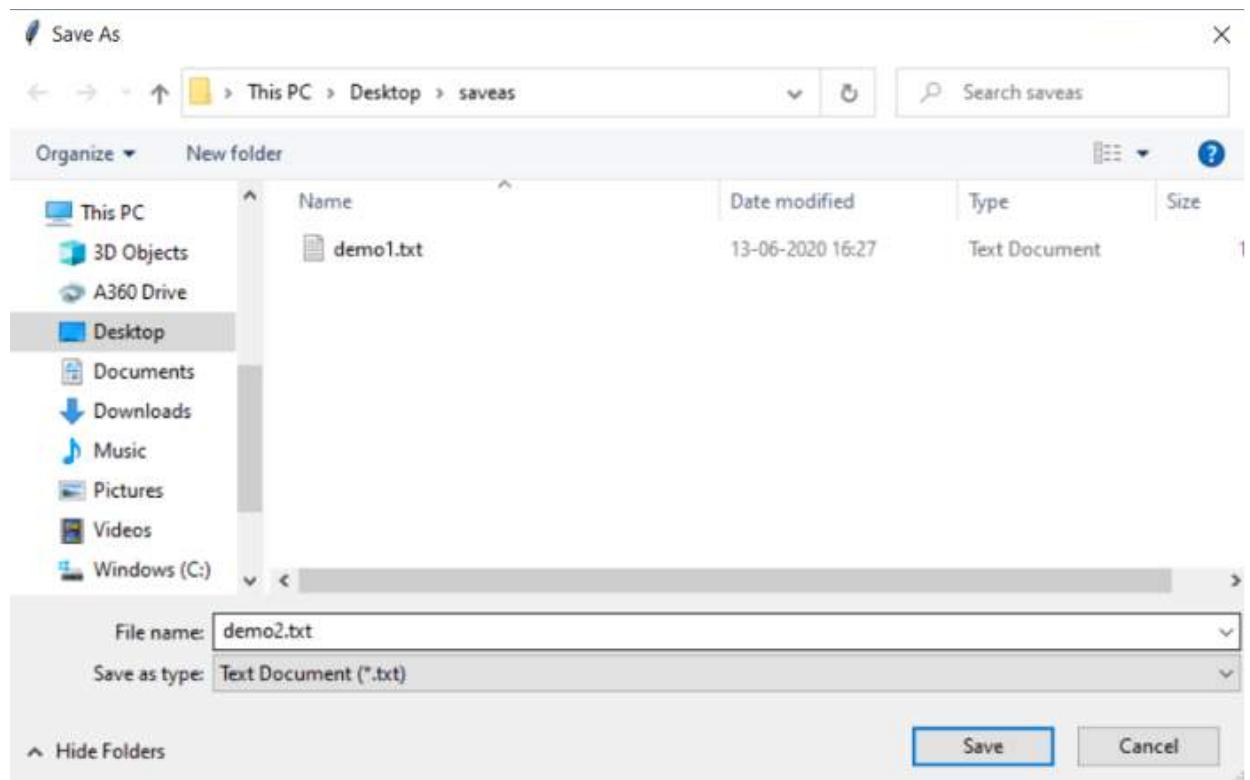


Figure 9.17: Output display on creation of a new file demo2.txt

The data inside the above file is as shown in [Figure 9.18](#):

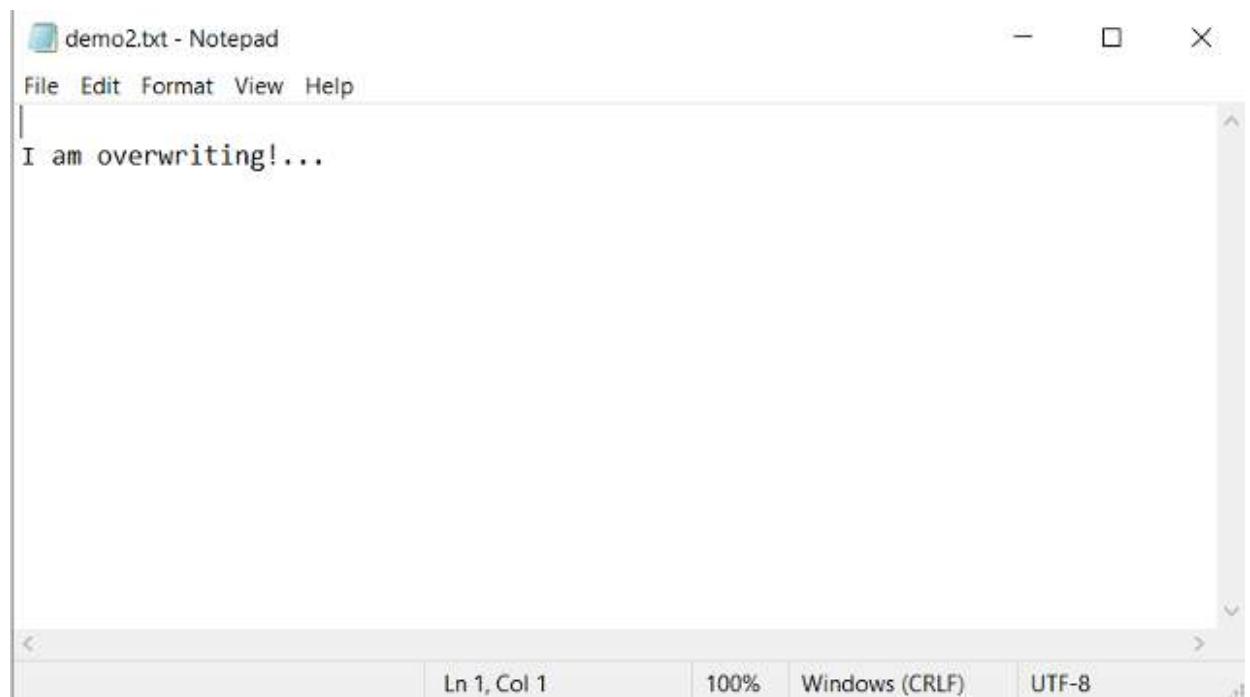


Figure 9.18: Output depicting the content inside demo2.txt file

**Note: The preceding code is covered in Program Name:
Chap9_Example6.py**

We can also provide a dialog box for color selection, as shown:

```
from tkinter import *
from tkinter.colorchooser import *

def myclickme():
    (my_rgb, mycolor) = askcolor(title = 'Please choose
your color') # tuple will be returned
    print(my_rgb)
    print(mycolor)
    myroot.configure(background=mycolor)

myroot = Tk()
myroot.title('ColorPicker')

mybtn1 = Button(myroot, text="Choose
Color", command=myclickme)
mybtn1.pack()
myroot.geometry("300x300")

myroot.mainloop()
```

Output:

Figure 9.19 shows the default output:

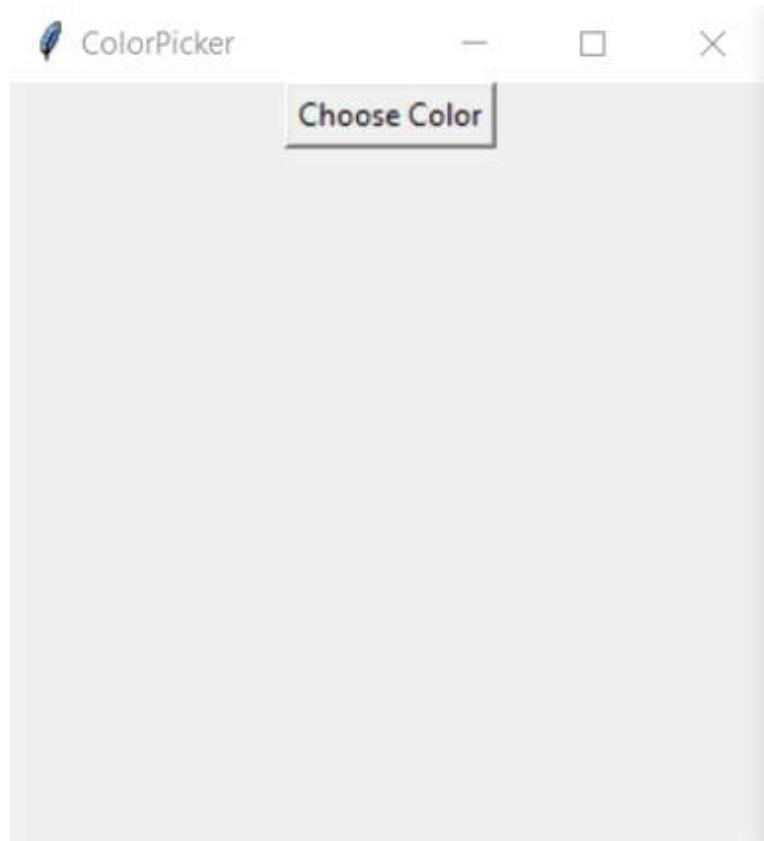


Figure 9.19: Default output

Figure 9.20 shows the output when the **Choose Color** button is clicked:

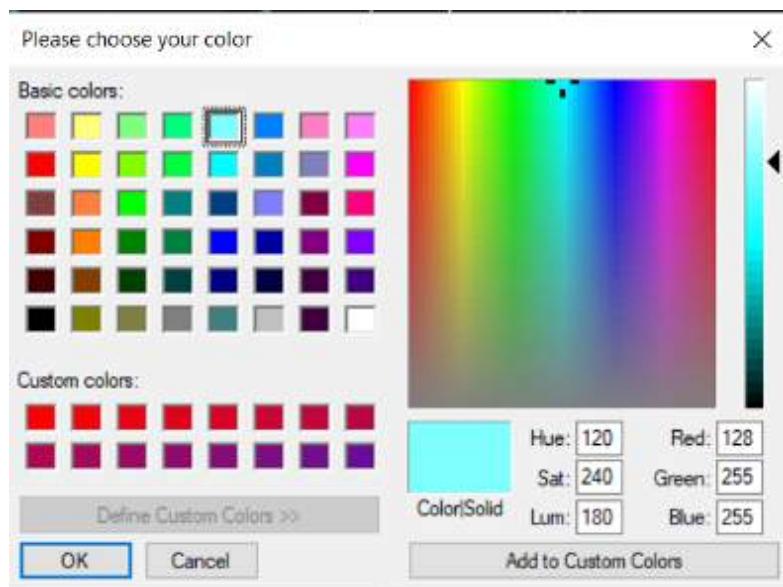


Figure 9.20: Output When Choose Color button is clicked

Figure 9.21 shows the output display of change of background color on pressing **OK** button:

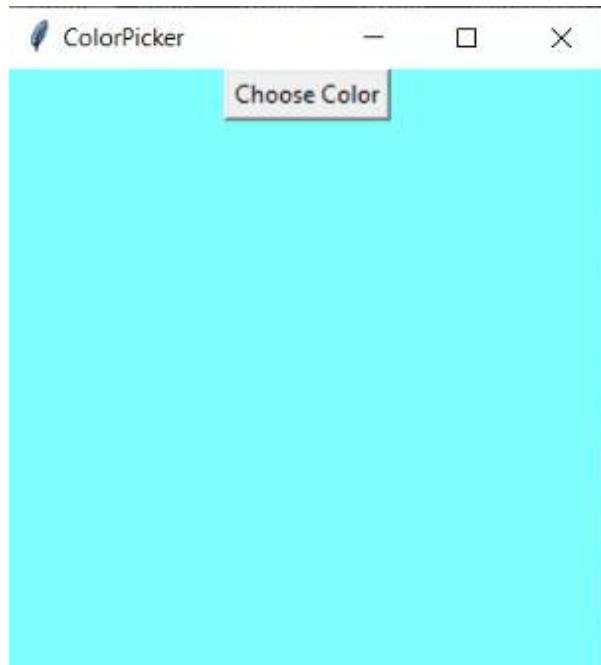


Figure 9.21: Output display of change of background color on pressing Ok button

Output at the console:

Refer to *Figure 9.22*:

```
(128.5, 255.99609375, 255.99609375)
#80ffff
```

Figure 9.22: Output at the console

**Note: The preceding code is covered in Program Name:
Chap9_Example7.py**

So, we can say that on clicking the **OK** button, a tuple is returned, which consists of a color value in RGB format and hexadecimal format as displayed. Moreover, we have changed the background color of the frame with the returned color value.

Conclusion

In this chapter, we learned that tkinter's handling of file selection is a rather straightforward process. To open, save, and browse files, we can utilize a number of the functions offered by the tkinter **filedialog** module. The **askopenfilename()** and **asksaveasfilename()** functions are the two that are most frequently utilized. With the use of these functions, a user can choose a file from their computer and select the path of that file.

Other functions that can be used to filter files, get information about files, and create custom file dialogs, are also provided by the **filedialog** module. In the end, we explored about prompt display of color selection dialog box by the user.

Points of remember

- The **filedialog** module is used to open, save and filter files. Moreover, with this module, we can get information about files and can create custom file dialogs.
- The **askopenfilename()** function allows selecting a file to open by the user.
- The **asksaveasfilename()** function allows selecting a file to save by the user.
- The **askdirectory()** function will return the directory path as a string and displays only directories.
- The **askopenfile()** function will return a file handle object and allows only the selection of existing files.
- The **askopenfilenames()** function will return file paths as list of strings and allows multiple selections.
- The **asksaveasfile** will return a file handle object and allows new file creations and confirmation on existing files to be prompted.

Questions

1. How is the process of opening and saving the file executed in Tkinter? Explain.
2. How is the file dialog window accessed in Tkinter? Explain in detail.
3. Write short notes on the following:
 - a. Askdirectory
 - b. Askopenfile
 - c. Askopenfilename
 - d. Askopenfilenames
 - e. Asksaveasfile
 - f. asksaveasfilename
4. Which widget is used to provide access for opening, closing, and saving the file? Explain the widget in detail.

CHAPTER 10

Getting Widget Information and Trace in tkinter

Introduction

Many a times, it is necessary to get to know the widget information and traces in a Python application. The widget's size, position, and condition are among the first details about which we can know. The widget may be displayed differently or controlled using this information. Additionally, it enables us to trace the changes made to the widget, such as when its text or state is altered. Other widgets or other actions may be updated using this information. Moreover, the behavior of the widget can be customized based on its current value or state by using trace. So, in this chapter, we shall learn about how to get widget information and explore various trace methods using tkinter library.

Structure

In this chapter, we will discuss the following topics:

- Getting widget information
- Trace in tkinter

Objectives

After going through this chapter, the reader will learn about how to get widget information with the help of multiple examples. We will also deal

with different trace methods viz trace add, trace remove, trace info, and so on.

Getting widget information

There are a set of **winfo_methods** that gives access to the widget information. A few useful methods are as follows:

- **winfo_width()**: This method will return the widget width.
- **winfo_height()**: This method will return the widget height.

Refer to the following code:

```
from tkinter import *
myroot = Tk()
mycanvas = Canvas(myroot, width= 300, height = 350)
print("Before packing width is: " +
str(mycanvas.winfo_width()))
print("Before packing height is: " +
str(mycanvas.winfo_height()))
mycanvas.pack()
mycanvas.update()
print("After packing and updating width is: " +
str(mycanvas.winfo_width()))
print("After packing and updating height is: " +
str(mycanvas.winfo_height()))
```

Output:

```
Before packing width is: 1
Before packing height is: 1
After packing width is: 304
After packing height is: 354
```

**Note: The preceding code is covered in Program Name:
Chap10_Example1.py**

- `winfo_children()`: This method will return a child widget's list.

```
from tkinter import *
myroot = Tk()
myroot.geometry('300x160')
mye1 = Entry(myroot)
mye1.pack(pady = 10)

mybtn1 = Button(myroot, text = 'Button')
mybtn1.pack(pady = 10)

myl1 = Label(myroot, text = 'Label')
myl1.pack(pady = 10)
# we are iterating on each child widget of the parent
window and disabling them
for loop in myroot.winfo_children():
    loop.config(state='disabled')

myroot.mainloop()
```

Output:

The output can be seen in *Figure 10.1*:

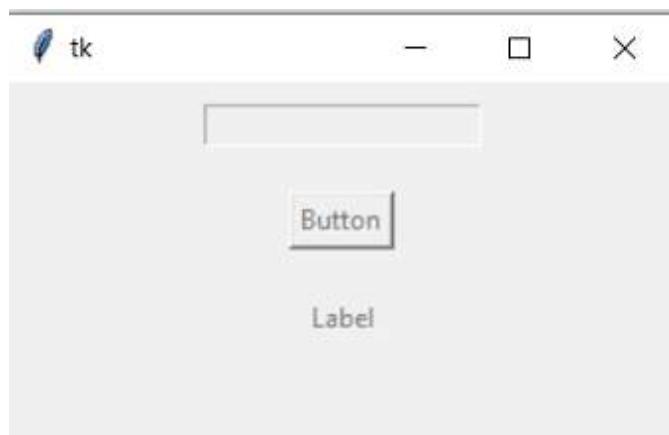


Figure 10.1: Output

**Note: The preceding code is covered in Program Name:
Chap10_Example2.py**

- **winfo_geometry()**: This method will return widget size and location.

```
from tkinter import *
myroot = Tk()
myroot.geometry('300x160+150+300')

myroot.update()
print(myroot.winfo_geometry())

myroot.mainloop()
```

Output at the console:

300x160+150+300

Output position of the window at the monitor screen

Where 300 is the width, 160 is the height, and (150 shifted on x axis, 300 shifted on y axis)

Refer to [*Figure 10.2*](#):

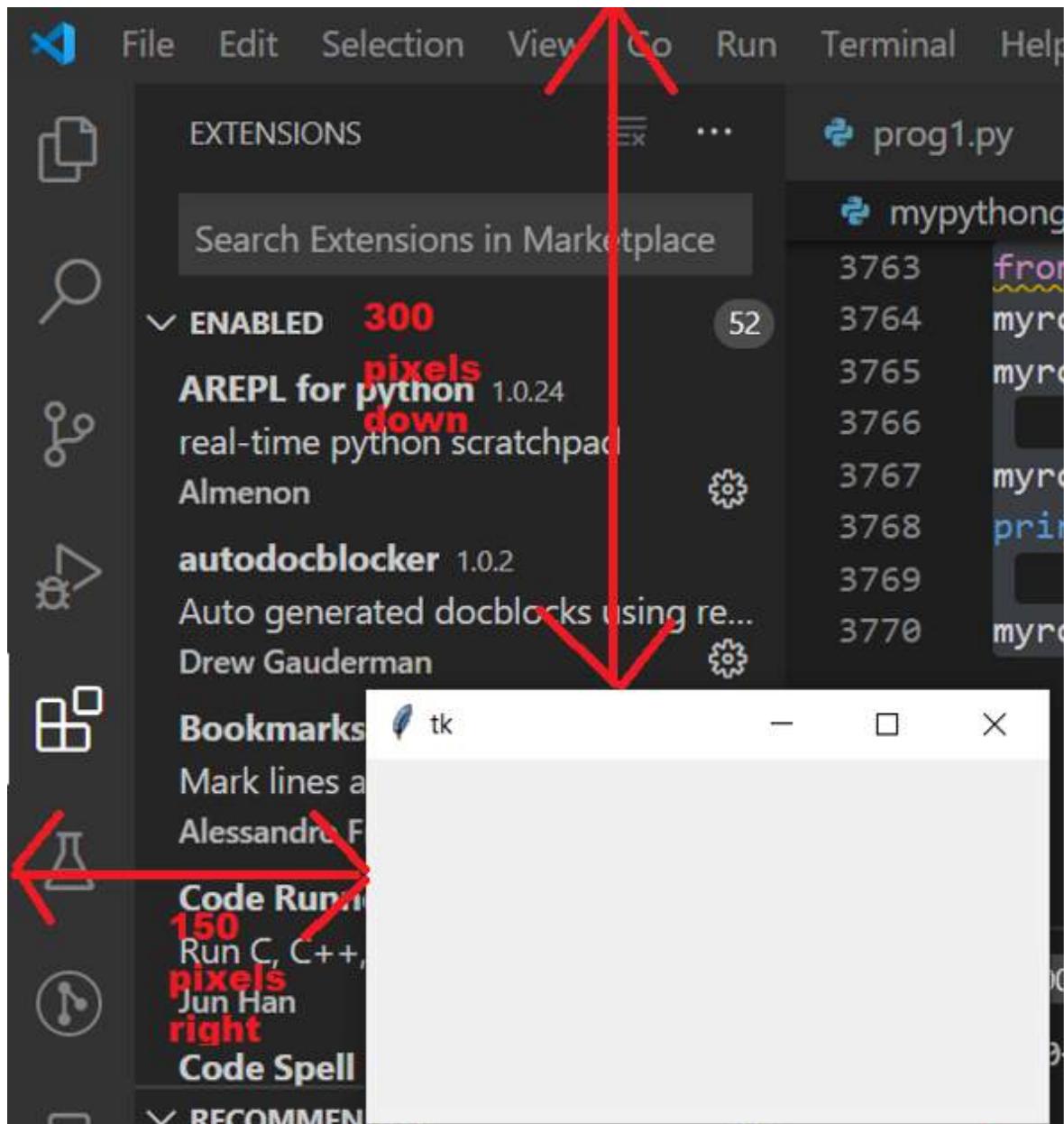


Figure 10.2: Output

Note: The preceding code is covered in Program Name:
Chap10_Example3.py

- `winfo_ismapped()`: This method will determine whether the widget is mapped or not, meaning that it has been added to the layout using a `pack()` and `grid()`.

```
from tkinter import *
```

```
myroot = Tk()

def myforget(widget):
    widget.forget()
    print(f"Is {widget['text']} mapped after calling
forget method ? = ",
bool(widget.winfo_ismapped()))

def myretrieve(widget):
    widget.pack()
    # will check if the widget exists or not
    print(f"Is {widget['text']} mapped after widget
retrieval ? = ",
bool(widget.winfo_exists()))

mybtn1 = Button(myroot, text = "IamButton1", bg =
'LightBlue')
mybtn1.pack(pady = 10)
# Making widget invisible
mybtn2 = Button(myroot, text = "Button2", command = lambda
: myforget(mybtn1), bg = 'LightGreen')
mybtn2.pack(pady = 10)

# Retrieving the widget
mybtn3 = Button(myroot, text = "Button3", command = lambda
: myretrieve(mybtn1), bg = 'LightPink')
mybtn3.pack(pady = 10)

myroot.geometry('300x200')

myroot.mainloop()
```

Output:

Refer to *Figure 10.3*:

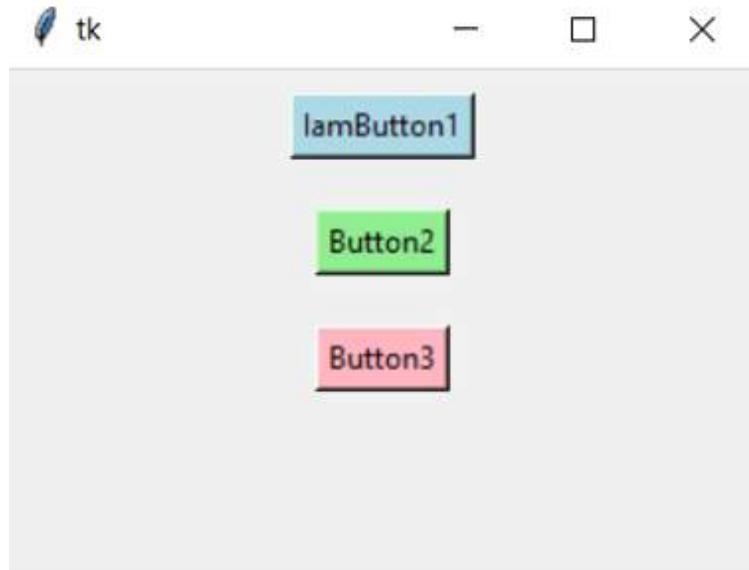


Figure 10.3: Default output of Chap10_Example4.py

Refer to *Figure 10.4*:

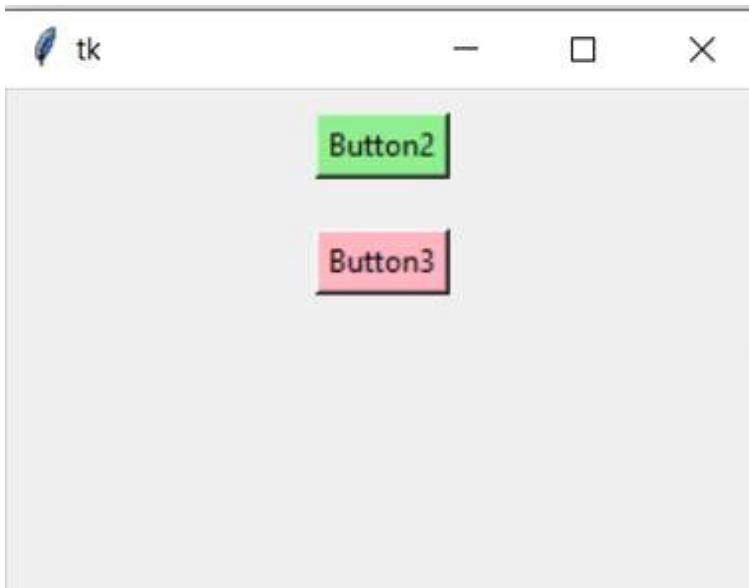


Figure 10.4: Output when Button2 is clicked

Refer to *Figure 10.5*:

```
$ python mypythonguiprog.py  
Is IamButton1 mapped after calling forget method ? = False  
[]
```

Figure 10.5: Output at the console when Button2 is clicked

Refer to [Figure 10.6](#):



Figure 10.6: Output when Button3 is clicked

Refer to [Figure 10.7](#):

```
$ python mypythonguiprog.py  
Is IamButton1 mapped after calling forget method ? = False  
Is IamButton1 mapped after widget retrieval ? = True  
[]
```

Figure 10.7: Output at the console when Button3 is clicked

**Note: The preceding code is covered in Program Name:
Chap10_Example4.py**

- **winfo_x()**: This method will get the x coordinate of the widget's top left corner.
- **winfo_y()**: This method will get the y coordinate of the widget's top left corner.

Refer to the following code:

```
from tkinter import *
myroot = Tk()

def myx_y_func():
    print(mybtn2.winfo_x())
    print(mybtn2.winfo_y())

mybtn2 = Button(myroot, text = "Button", command =
myx_y_func, bg = 'LightGreen')
mybtn2.pack(pady = 10)
myroot.geometry('300x100')

myroot.mainloop()
```

Output when the Button is clicked:

Refer to *Figure 10.8*:

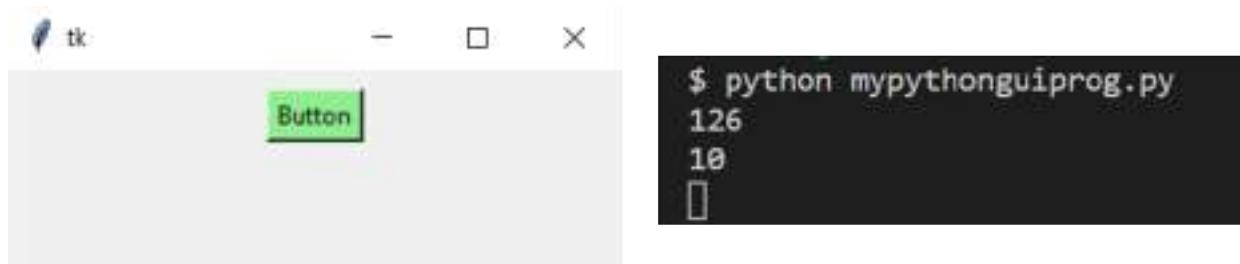


Figure 10.8: Output

**Note: The preceding code is covered in Program Name:
Chap10_Example5.py**

Trace in tkinter

To track variables in Python, variable wrappers are created by tkinter by attaching an ‘observer’ callback to the variable. Variable classes such as

BooleanVar, **StringVar**, **DoubleVar**, **IntVar** for Boolean, string, double, and integer values respectively can be registered to an observer, which gets triggered whenever the variable value is accessed. Until an observer is explicitly deleted, it remains active. The callback function associated with an observer takes 3 arguments, which are as follows:

- Mode of access
- The tkinter variable index in case it is an array else an empty string.
- The tkinter variable name.

Let us now study the various trace methods.

trace_add()

This method will replace **trace_variable()** method. It will add an observer to a name and return the callback function name whenever the value is accessed.

The syntax is as follows:

trace_add(self, mode, callback_name)

where,

- The **mode** parameter can be one of ‘array’, ‘read’, ‘write’, ‘unset’, or a list or tuple of such strings.
- **callback_name** parameter is a callback function name that is to be registered on the tkinter variable.

trace_remove()

This method will replace the **trace_vdelete()** method. It will unregister an observer and initially while registering the observer through the **trace_add()** method, it will return the callback name.

The syntax is as follows:

trace_remove(self, mode, callback_name)

where,

- The **mode** parameter can be one of ‘array’, ‘read’, ‘write’, ‘unset’, or a list or tuple of such strings.
- **callback_name** parameter is a callback function name that is to be registered on the tkinter variable.

trace_info()

This method will replace **trace_vinfo()** and the trace method, which returns a callback name and is used to find the callback name which is to be deleted. The argument here is the tkinter variable itself.

The syntax is as follows:

trace_info(self)

Refer to the following code:

```
from tkinter import *
myroot = Tk()
myroot.title('Trace_add')
myroot.geometry('300x200')
my_value = StringVar()

mybtn1 = Button(myroot, textvariable = my_value, bg =
'LightBlue')
mybtn1.pack(padx = 10, pady = 10)
mye1 = Entry(myroot, textvariable = my_value, bg =
'LightGreen')
mye1.pack(padx = 10, pady = 10)
# defining the callback function (observer)
def my_associatedcallback(var, indx, mode):
    print("Traced variable {}: {}".format(my_value.get()))

# registering the observer
```

```
my_value.trace_add('write', my_associatedcallback)  
myroot.mainloop()
```

Output initially:

Refer to *Figure 10.9*:

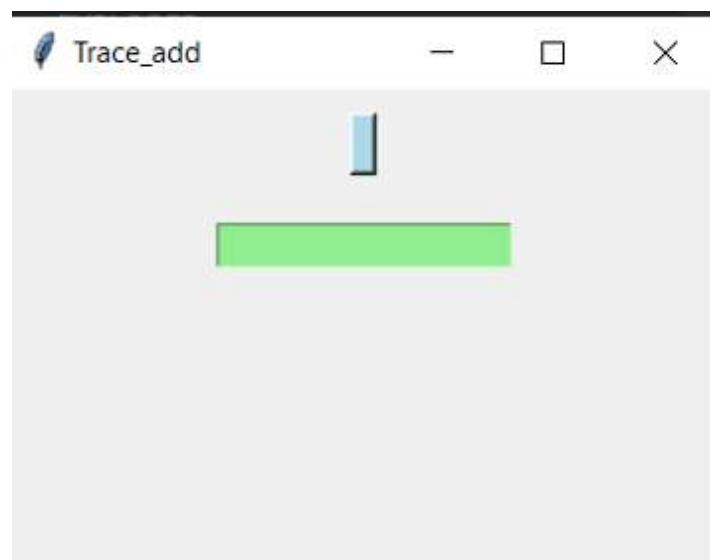


Figure 10.9: Default output of Chap10_Example6.py

Refer to *Figure 10.10*:

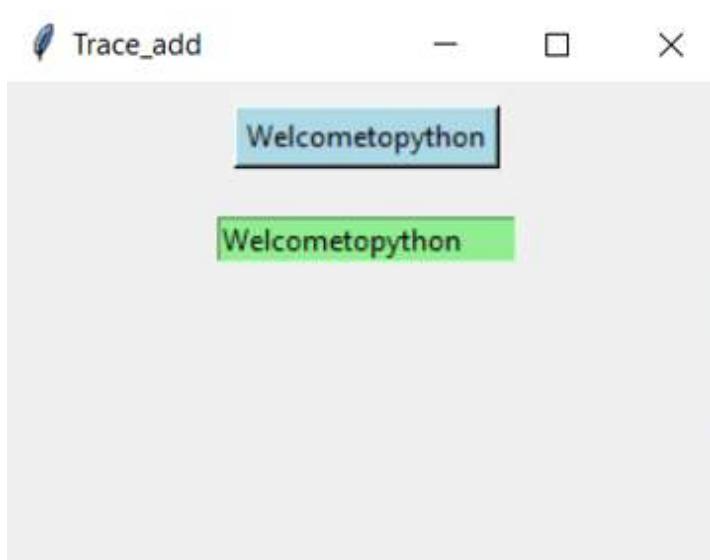


Figure 10.10: Output when the data is entered in the Entry widget

Refer to *Figure 10.11*:

```
$ python mypythonguiproj.py
Traced variable W:
Traced variable We:
Traced variable Wel:
Traced variable Welc:
Traced variable Welco:
Traced variable Welcom:
Traced variable Welcome:
Traced variable Welcomet:
Traced variable Welcometo:
Traced variable Welcometop:
Traced variable Welcometopy:
Traced variable Welcometopyt:
Traced variable Welcometopyth:
Traced variable Welcometopytho:
Traced variable Welcometopython:
[]
```

Figure 10.11: Output at the console when user entering the text

**Note: The preceding code is covered in Program Name:
Chap10_Example6.py**

In the above code, the callback function will be triggered whenever the widget text changes and will return a string “Traced Variable”, that is, the moment the text in the entry widget is written, the text in the Button widget will be written simultaneously.

Conclusion

In this chapter, we learned that understanding tkinter’s widget information and trace information is crucial for producing good Graphical User Interfaces. We can access widget information to learn more about a widget, including its size, location, and state. The widget may be displayed differently or controlled using this information. We may track changes to the widget, such as when its text is changed or its status is altered, using the trace information. Other widgets or other actions may be updated using this

information. Finally, we explored about different trace methods viz **trace_add()**, **trace_remove()** and **trace_info()**.

Points to remember

- We can access widget information to get information about a widget, including its size, location, and state.
- The **winfo_width()** and **winfo_height()** methods can be used to obtain a widget's width and height, which can subsequently be used to adjust the widget's size.
- We may track changes to the widget, such as when its text is changed or its status is modified, using the trace information.
- Widget information can be used to change how the widget is displayed or controlled.
- We can update other widgets or carry out additional tasks using trace information.
- To obtain widget information, make use of **winfo_** methods.
- The **trace()** method can be used to track changes made to a widget.
- A trace callback can be added to a widget using the **trace_add()** method, and information about the trace callbacks connected to a widget can be obtained using the **trace_info()** method.
- To remove a trace callback, use the **trace_remove()** method.

Questions

1. How can we get widget information in tkinter? Explain in detail.
2. What is the meaning of trace in tkinter? Explain in detail.
3. Explain the trace methods used in the tkinter GUI application.
4. Explain the following:
 - a. **trace_add()**

- b. trace_remove()
 - c. trace_info()
5. How does tkinter GUI interact with Sqlite3 database? Explain the entire process in detail.

CHAPTER 11

UserLogin Project in tkinter GUI Library with sqlite3 Database

Introduction

In this chapter, we shall see a **Graphical User Interface (GUI)** application by creating a UserLogin project with sq-lite3 database. The code shall be explained along with the desired output. We will also learn how a GUI can be used to interact with a sqlite3 database.

Structure

In this chapter, we will discuss the following topics:

- GUI interaction with sqlite3 database
- Displaying a GUI application

Objectives

By the end of this chapter, the reader will know about GUI interaction with sqlite3 database. We will be learning how to connect any GUI program with sqlite3 database, and here we are creating a small GUI application using tkinter library. We shall see different steps of interacting with sqlite3 database, starting with importing the module, connecting to the database, creating a cursor object, then executing queries, committing the changes and finally closing the connection.

GUI interaction with sqlite3 database

To use GUI interaction with sqlite3 database, we will follow the given steps:

1. We will be using an **import** statement to our Python program:

```
import sqlite3
```

2. The next step is to connect to the database. We will be using **sqlite3.connect()** function by passing the file name to open or create it.

```
mydb = sqlite3.connect('demo.db')
```

3. Once the connection is done, we can create a cursor object and then call the **execute()** method to perform SQL commands:

```
mycr = mydb.cursor()
```

4. Then create a table, using:

```
mycr.execute('create table login(UNAME text,  
UPASS text)')
```

5. Commit the changes, that is, save the changes:

```
mydb.commit()
```

6. Close the connection:

```
mydb.close()
```

If we close the database connection without calling commit, the changes will be lost.

Displaying a GUI application

Let us now see a Python code to display a GUI application of UserLogin Project with tkinter library, along with connectivity to sqlite3 database. You will get a glimpse of GUI code with connectivity to sqlite3 database.

Practice the following code and follow the steps as instructed:

```
from tkinter import *  
import sqlite3  
  
from tkinter import messagebox
```

```
from tkinter import ttk
myroot = Tk()
myroot.geometry('600x400')
myroot.resizable(0,0)
myroot.title('Home Page')
myframe55 = None

def myscreen():
    myntb = ttk.Notebook()
    def mydemo(a1):
        print(myntb.index('current')) # getting tab
positions
        if myntb.index('current') == 5:
            myhome()

    myntb.bind('<<NotebookTabChanged>>', mydemo)
    myntb.place(x=0,y=0,width = 600, height = 400)

    bginsertion(myntb)
    myshowall(myntb)
    mysearch(myntb)
    myupdate(myntb)
    mydelete(myntb)
    mylogout(myntb)
def bginsertion(ntb):
    myf4 = Frame(bg = 'LightBlue')
    ntb.add(myf4,text='MyInsert')

m = StringVar()
```

```
n = StringVar()
o = StringVar()
p = StringVar()
q = StringVar()

myl1 = Label(myf4, font = ('Calibri',15),text = 'Enter
Roll No.', bg = 'LightBlue', fg = 'Red')
myl1.place(x=150,y=50)
mye1 = Entry(myf4, font = ('Calibri',15), textvariable
= m)
mye1.place(x = 300, y = 50, width = 100)

myl2 = Label(myf4, font = ('Calibri',15),text = 'Enter
Name', bg = 'LightBlue', fg = 'Red')
myl2.place(x=150,y=100)
mye2 = Entry(myf4, font = ('Calibri',15), textvariable
= n)
mye2.place(x = 300, y = 100, width = 100)

myl3 = Label(myf4, font = ('Calibri',15),text = 'Enter
Phy.', bg = 'LightBlue', fg = 'Red')
myl3.place(x=150,y=150)
mye3 = Entry(myf4, font = ('Calibri',15), textvariable
= o)
mye3.place(x = 300, y = 150, width = 100)

myl4 = Label(myf4, font = ('Calibri',15),text = 'Enter
Chem.', bg = 'LightBlue', fg = 'Red')
myl4.place(x=150,y=200)
```

```

mye4 = Entry(myf4, font = ('Calibri',15), textvariable
= p)
mye4.place(x = 300, y = 200, width = 100)

myl5 = Label(myf4, font = ('Calibri',15),text = 'Enter
Maths.', bg = 'LightBlue', fg = 'Red')
myl5.place(x=150,y=250)
mye5 = Entry(myf4, font = ('Calibri',15), textvariable
= q)
mye5.place(x = 300, y = 250, width = 100)

def mydatainsertion():
    mydb = sqlite3.connect('mydemo.db')
    my_cursor = mydb.cursor()
    my_cursor.execute("insert into ins
values('"+m.get()+"','"+n.get()+"','"+o.get()+"','"+p.get(
)+"','"+q.get()+"')")
    mydb.commit()
    mydb.close()
    messagebox.showinfo('Title','Data Inserted')
    m.set('')
    n.set('')
    o.set('')
    p.set('')
    q.set('')
    myshowalldata(myframe55)

    mybtn = Button(myf4, font = ('Calibri',15),text =
'Insert Data', bg = 'LightBlue', fg = 'Red', command =
mydatainsertion)

    mybtn.place(x = 250, y = 300, width = 100, height =

```

```
30)

def myshowall(ntb):
    myf5 = Frame(bg = 'LightBlue')
    ntb.add(myf5, text='MyShowAll')
    global myframe55
    myframe55 = myf5
    myshowalldata(myf5)

def myshowalldata(myf5):
    # for deletion addition
    for loop in myf5.winfo_children(): # a list of all
        widgets are returned
        loop.destroy()

    myu1 = Label(myf5, font = ('Arial',12), text = 'Roll
No.',bg = 'LightBlue', fg = 'Red')
    myu1.place(x=0,y=0, width = 120)

    myu2 = Label(myf5, font = ('Arial',12), text =
'Name',bg = 'LightBlue', fg = 'Red')
    myu2.place(x=120,y=0, width = 120)

    myu3 = Label(myf5, font = ('Arial',12), text =
'Phy.',bg = 'LightBlue', fg = 'Red')
    myu3.place(x=240,y=0, width = 120)

    myu4 = Label(myf5, font = ('Arial',12), text =
'Chem.',bg = 'LightBlue', fg = 'Red')
    myu4.place(x=360,y=0, width = 120)
```

```

myu5 = Label(myf5, font = ('Arial',12), text =
'Maths',bg = 'LightBlue', fg = 'Red')
myu5.place(x=480,y=0, width = 120)

db = sqlite3.connect('mydemo.db')
cr = db.cursor()
r = cr.execute("select * from ins ")
x = 50
y = 60
for loop in r:
    Label(myf5,text = loop[0], font = ('Arial',12),bg
= 'LightBlue', fg = 'Red').place(x=x,y=y)
    x += 120
    Label(myf5,text = loop[1], font = ('Arial',12),bg
= 'LightBlue', fg = 'Red').place(x=x,y=y)
    x += 120
    Label(myf5,text = loop[2], font = ('Arial',12),bg
= 'LightBlue', fg = 'Red').place(x=x,y=y)
    x += 120
    Label(myf5,text = loop[3], font = ('Arial',12),bg
= 'LightBlue', fg = 'Red').place(x=x,y=y)
    x += 120
    Label(myf5,text = loop[4], font = ('Arial',12),bg
= 'LightBlue', fg = 'Red').place(x=x,y=y)
    y += 60
    x = 50
def mysearch(ntb):
    myf6 = Frame(bg = 'LightBlue')
    ntb.add(myf6,text='MySearch')

```

```
s1 = StringVar()

myu1 = Label(myf6, font = ('Arial',12), text = 'Roll
No.',bg = 'LightBlue', fg = 'Red')
myu1.place(x=100,y=50, width = 120)

mye1 = Entry(myf6, font = ('Calibri',15), textvariable
= s1)
mye1.place(x = 200, y = 50, width = 100)

def searched():
    db = sqlite3.connect('mydemo.db')
    cr = db.cursor()
    r = cr.execute("select * from ins where URNO
='"+s1.get()+"' ")
    for loop in r:
        myl1 = Label(myf6, text = "Name is: ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl1.place(x=200,y=100)

        myl2 = Label(myf6, text = loop[1], font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl2.place(x=350,y=100)

        myl1 = Label(myf6, text = "Phy : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
```

```
myl1.place(x=200,y=150)

myl2 = Label(myf6, text = loop[2], font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl2.place(x=350,y=150)

myl1 = Label(myf6, text = "Chem : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl1.place(x=200,y=200)

myl2 = Label(myf6, text = loop[3], font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl2.place(x=350,y=200)

myl1 = Label(myf6, text = "Maths : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl1.place(x=200,y=250)

myl2 = Label(myf6, text = loop[4], font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl2.place(x=350,y=250)
break
else:
    messagebox.showinfo('Title','Roll No. absent')
    myl11 = Label(myf6, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
    myl11.place(x=200,y=100,width=300)
```

```
    myl12 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl12.place(x=350,y=100,width=300)
```

```
    myl13 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl13.place(x=200,y=150,width=300)
```

```
    myl14 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl14.place(x=350,y=150,width=300)
```

```
    myl15 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl15.place(x=200,y=200,width=300)
```

```
    myl16 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl16.place(x=350,y=200,width=300)
```

```
    myl17 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl17.place(x=200,y=250,width=300)
```

```
    myl18 = Label(myf6, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl18.place(x=350,y=250)  
db.commit()
```

```
db.close()

mybtn = Button(myf6, text = 'Search', font = ('Calibri',15), command = searched)
mybtn.place(x=320,y=50, width = 100,height = 30)

def myupdate(ntb):
    myf7 = Frame(bg = 'LightBlue')
    ntb.add(myf7,text='MyUpdate')

s2 = StringVar()

myu1 = Label(myf7, font = ('Arial',12), text = 'Roll
No.',bg = 'LightBlue', fg = 'Red')
myu1.place(x=100,y=50, width = 120)

mye1 = Entry(myf7, font = ('Calibri',15), textvariable
= s2)
mye1.place(x = 200, y = 50, width = 100)

def updated():
    db = sqlite3.connect('mydemo.db')
    cr = db.cursor()
    r = cr.execute("select * from ins where URNO
='"+s2.get()+"' ")
    for loop in r:
        s3 = StringVar()
        s4 = StringVar()
```

```
s5 = StringVar()
s6 = StringVar()

myl1 = Label(myf7, text = "Name is: ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl1.place(x=200,y=100)

myl2 = Entry(myf7, font = ('Calibri',15), bg =
'LightBlue', fg = 'Red', textvariable = s3)
myl2.insert(0,loop[1])
myl2.place(x=350,y=100)

myl1 = Label(myf7, text = "Phy : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl1.place(x=200,y=150)

myl2 = Entry(myf7, font = ('Calibri',15), bg =
'LightBlue', fg = 'Red', textvariable = s4)
myl2.insert(0,loop[2])
myl2.place(x=350,y=150)

myl1 = Label(myf7, text = "Chem : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
myl1.place(x=200,y=200)

myl2 = Entry(myf7, font = ('Calibri',15), bg =
'LightBlue', fg = 'Red', textvariable = s5)
myl2.insert(0,loop[3])
myl2.place(x=350,y=200)
```

```

        myl1 = Label(myf7, text = "Maths : ", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl1.place(x=200,y=250)

        myl2 = Entry(myf7, font = ('Calibri',15), bg =
'LightBlue', fg = 'Red', textvariable = s6)
        myl2.insert(0,loop[4])
        myl2.place(x=350,y=250)

def updatedata2():
    mydb = sqlite3.connect('mydemo.db')
    my_cursor = mydb.cursor()
    my_cursor.execute("update ins set UNAME =
"+s3.get()+"", UPHY =" "+s4.get()+"", UCHE
=" "+s5.get()+"", UMATHS =" "+s6.get()+" WHERE URNO
=" "+s2.get()+" ")
    mydb.commit()
    mydb.close()
    messagebox.showinfo('Title','Data
Updated')
    s3.set('')
    s4.set('')
    s5.set('')
    s6.set('')
    myshowalldata(myframe55)

mybtn = Button(myf7, text = 'Update', font =
('Calibri',15), command = updatedata2)
mybtn.place(x=250,y=325, width = 100,height =
30)

```

```
        break
    else:
        messagebox.showinfo('Title','Roll No. absent')
        myl11 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl11.place(x=200,y=100,width=300)

        myl12 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl12.place(x=350,y=100,width=300)

        myl13 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl13.place(x=200,y=150,width=300)

        myl14 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl14.place(x=350,y=150,width=300)

        myl15 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl15.place(x=200,y=200,width=300)

        myl16 = Label(myf7, text = "", font =
('Calibri',15), bg = 'LightBlue', fg = 'Red')
        myl16.place(x=350,y=200,width=300)
```

```
    myl17 = Label(myf7, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl17.place(x=200,y=250,width=300)
```

```
    myl18 = Label(myf7, text = "", font =  
('Calibri',15), bg = 'LightBlue', fg = 'Red')  
    myl18.place(x=350,y=250)  
db.commit()  
db.close()
```

```
mybtn = Button(myf7, text = 'Retrieve', font =  
('Calibri',15), command = updated)  
mybtn.place(x=320,y=50, width = 100,height = 30)
```

```
def mydelete(ntb):  
    myf8 = Frame(bg = 'LightBlue')  
    ntb.add(myf8,text='MyDelete')
```

```
s1 = StringVar()
```

```
myu1 = Label(myf8, font = ('Arial',12), text = 'Roll  
No.',bg = 'LightBlue', fg = 'Red')  
myu1.place(x=100,y=50, width = 120)
```

```
mye1 = Entry(myf8, font = ('Calibri',15), textvariable  
= s1)  
mye1.place(x = 200, y = 50, width = 100)
```

```
def mydeletion():
    db = sqlite3.connect('mydemo.db')
    cr = db.cursor()
    r = cr.execute("delete from ins where URNO
= '"+s1.get()+"' ")
    messagebox.showinfo('Title','Data deleted')
    db.commit()
    db.close()
    myshowalldata(myframe55)
    s1.set('')

mybtn = Button(myf8, text = 'Delete', font =
('Calibri',15), command = mydeletion)
mybtn.place(x=320,y=50, width = 100,height = 30)
def mylogout(ntb):
    myf9 = Frame(bg = 'LightBlue')
    ntb.add(myf9,text='MyLogOut')
def mylogin():
    myf2 = Frame(bg = 'LightBlue')
    myf2.place(x=0,y=0, width = 600, height = 400)

d = StringVar()
e = StringVar()

myl1 = Label(myf2, text = 'Enter Name: ', bg =
'LightBlue', fg = 'Red')
myl1.place(x=200,y=100)
```

```
mye1 = Entry(myf2, font = ('Calibri',15), textvariable = d)
mye1.place(x = 300, y =100, width = 100, height = 20)

myl2 = Label(myf2, text = 'Enter Password: ', bg = 'LightBlue', fg = 'Red')
myl2.place(x=200,y=150)
mye2 = Entry(myf2, font = ('Calibri',15), textvariable = e)
mye2.place(x = 300, y =150, width = 100, height = 20)

def login1():
    db = sqlite3.connect('mydemo.db')
    cr = db.cursor()
    r = cr.execute("select * from regis where UNAME ='" +d.get() +"' AND UPASS ='" +e.get() +"' ")
    for loop in r:
        myscreen()
        break
    else:
        messagebox.showinfo('Title','Invalid user name and password')
    db.commit()
    db.close()

mybtn = Button(myf2, text = 'Login', font = ('Calibri',15), command = login1)
mybtn.place(x=250,y=200, width = 100,height = 30)
```

```
mybtn1 = Button(myf2, text = 'Home', font = ('Calibri',15), command = myhome)
mybtn1.place(x=20,y=350, width = 100,height = 30)

mybtn2 = Button(myf2, text = 'Registration', font = ('Calibri',15), command = myregis)
mybtn2.place(x=480,y=350, width = 120,height = 30)

def myregis():
    myf2 = Frame(bg = 'LightBlue')
    myf2.place(x=0,y=0, width = 600, height = 400)

    a = StringVar()
    b = StringVar()
    c = StringVar()

    myl1 = Label(myf2, text = 'Enter Name: ', bg = 'LightBlue', fg = 'Red')
    myl1.place(x=200,y=100)
    mye1 = Entry(myf2, font = ('Calibri',15), textvariable = a)
    mye1.place(x = 300, y =100, width = 100, height = 20)

    myl2 = Label(myf2, text = 'Enter Password: ', bg = 'LightBlue', fg = 'Red')
    myl2.place(x=200,y=150)
    mye2 = Entry(myf2, font = ('Calibri',15), textvariable = b)
    mye2.place(x = 300, y =150, width = 100, height = 20)
```

```
    myl3 = Label(myf2, text = 'Enter CN: ', bg = 'LightBlue', fg = 'Red')
    myl3.place(x=200,y=200)
    mye3 = Entry(myf2, font = ('Calibri',15), textvariable = c)
    mye3.place(x = 300, y =200, width = 100, height = 20)

def registering():
    mydb = sqlite3.connect('mydemo.db')
    my_cursor = mydb.cursor()
    my_cursor.execute("insert into regis
values('"+a.get()+"','"+b.get()+"','"+c.get()+"')")
    mydb.commit()
    mydb.close()
    messagebox.showinfo('Title','User Registered')
    a.set('')
    b.set('')
    c.set('')

    mybtn = Button(myf2, text = 'Register', font = ('Calibri',15), command = registering)
    mybtn.place(x=250,y=250, width = 120,height = 30)

    mybtn1 = Button(myf2, text = 'Home', font = ('Calibri',15), command = myhome)
    mybtn1.place(x=20,y=350, width = 100,height = 30)

    mybtn2 = Button(myf2, text = 'Login', font = ('Calibri',15), command = mylogin)
```

```
mybtn2.place(x=480, y=350, width = 120,height = 30)

def myhome():
    myf1 = Frame(bg = 'LightBlue')
    myf1.place(x=0,y=0, width = 600, height = 400)

    myb1 = Button(myf1, text = 'Login', command = mylogin)
    myb1.place(x=220,y=100,width = 100, height = 30)

    myb2 = Button(myf1, text = 'Register', command = myregis)
    myb2.place(x=330,y=100,width = 100, height = 30)

myhome()

myroot.mainloop()
```

**Note: The preceding code is covered in (Program Name:
Chap11_Example1.py)**

Then follow the given steps:

1. **Sqlite3 Database Structure:** The sqlite3 database structure can be seen in the following *Figure 11.1*:

Database Structure			
Create Table Create Index Print			
Name	Type	Schema	
Tables (2)			
ins		CREATE TABLE ins(URNO text, UNAME text, UPHY text, UCHE text, UMATHS text)	
URNO	text	"URNO" text	
UNAME	text	"UNAME" text	
UPHY	text	"UPHY" text	
UCHE	text	"UCHE" text	
UMATHS	text	"UMATHS" text	
regis		CREATE TABLE regis(UNAME text, UPASS text, UCN text)	
UNAME	text	"UNAME" text	
UPASS	text	"UPASS" text	
UCN	text	"UCN" text	
Indices (0)			
Views (0)			
Triggers (0)			

Figure 11.1: sqlite3 database structure

2. Sqlite3 Browse Data: The sqlite3 browse data can be seen in the following [Figure 11.2](#):

The screenshot shows the SQLite3 browser interface with two tables displayed:

Table: regis

	UNAME	UPASS	UCN
	Filter	Filter	Filter
1	Saurabh	1234	10001
2	Nilesh	4321	10002
3	a	1	10003

Table: ins

	URNO	UNAME	UPHY	UCHE	UMATHS
	Filter	Filter	Filter	Filter	Filter
1	1	Ram	88	90	100
2	3	Nilesh	89	92	99
3	4	Saurabh	92	100	100
4	5	Divya	100	90	92

Figure 11.2: sqlite3 browse data

3. Output when the program has started to run: The output can be seen in [*Figure 11.3*](#):

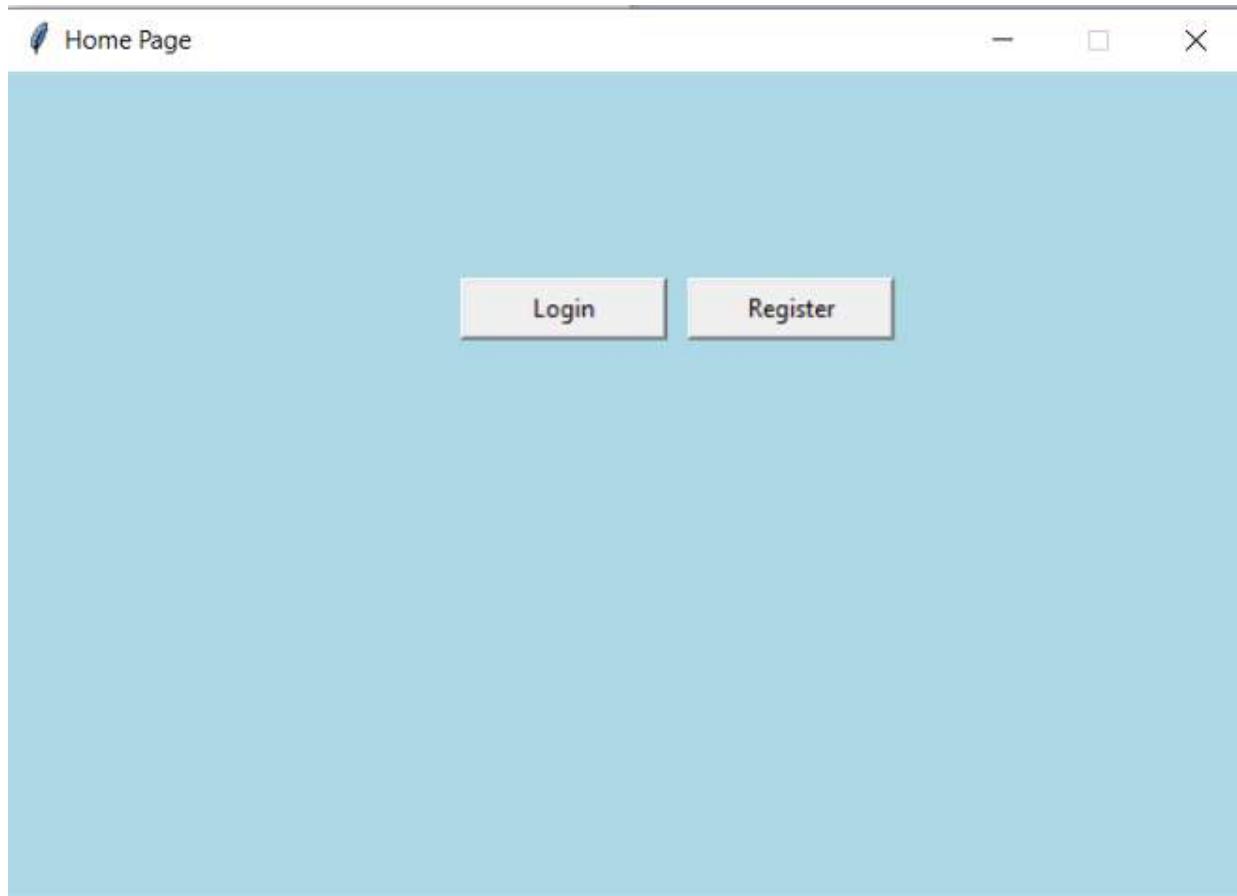


Figure 11.3: Output when the program starts running

4. **Output when Login button is clicked:** The output when login button is clicked, can be seen in [Figure 11.4](#):

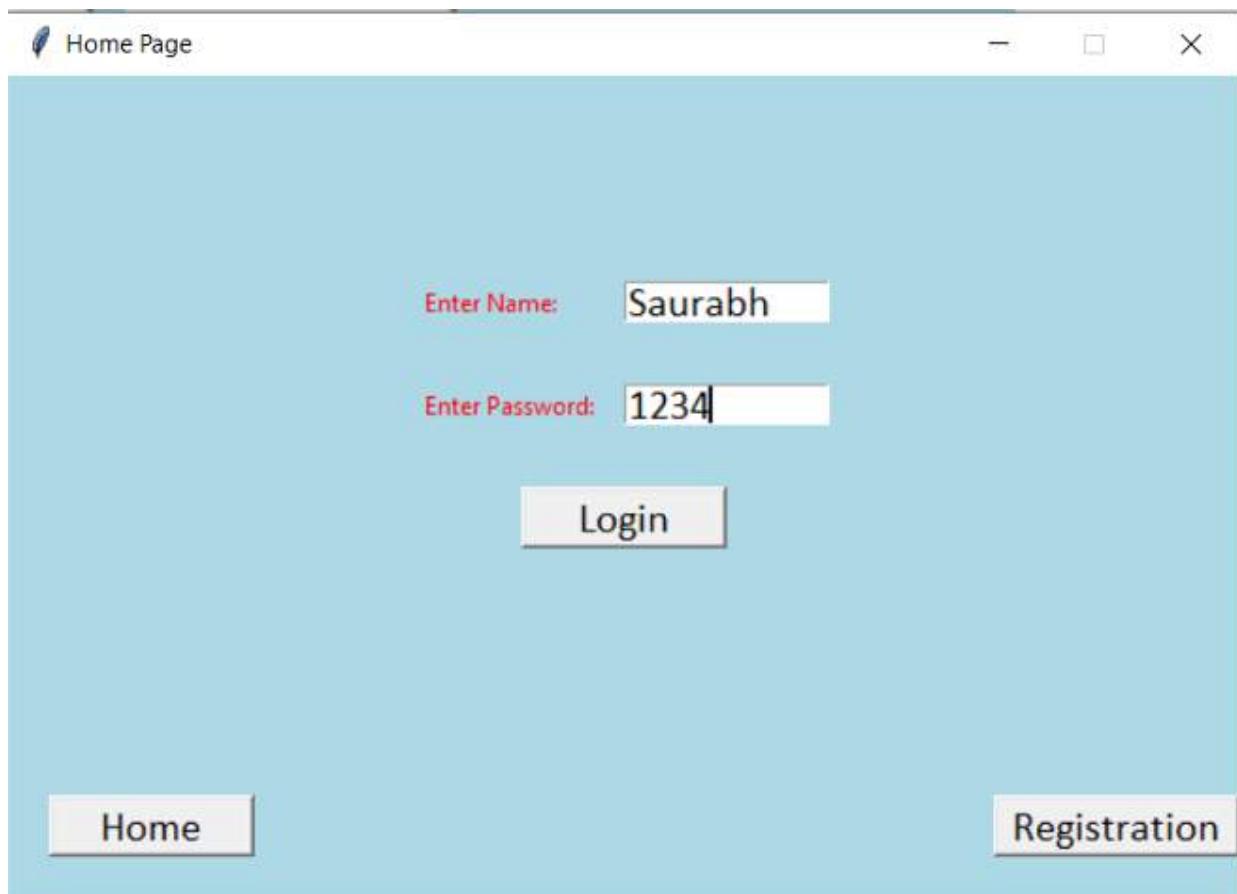


Figure 11.4: Output when Login button is clicked

5. On clicking the **Home** button, the user will navigate to the **Home Page** consisting of **Login** and **Register** button.
6. The user is requested to enter username and password and click on the **Login** button, as shown in *Figure 11.5*:

The screenshot shows a Windows application window titled "Home Page". The menu bar at the top includes "MyInsert", "MyShowAll", "MySearch", "MyUpdate", "MyDelete", and "MyLogOut". The main area contains five input fields labeled "Enter Roll No.", "Enter Name", "Enter Phy.", "Enter Chem.", and "Enter Maths.", each followed by a white input box. Below these fields is a red-bordered button labeled "Insert Data".

Figure 11.5: Insert data in Login page

7. In the **MyInsert** tab form, the user can insert the data, as shown in *Figure 11.6*:

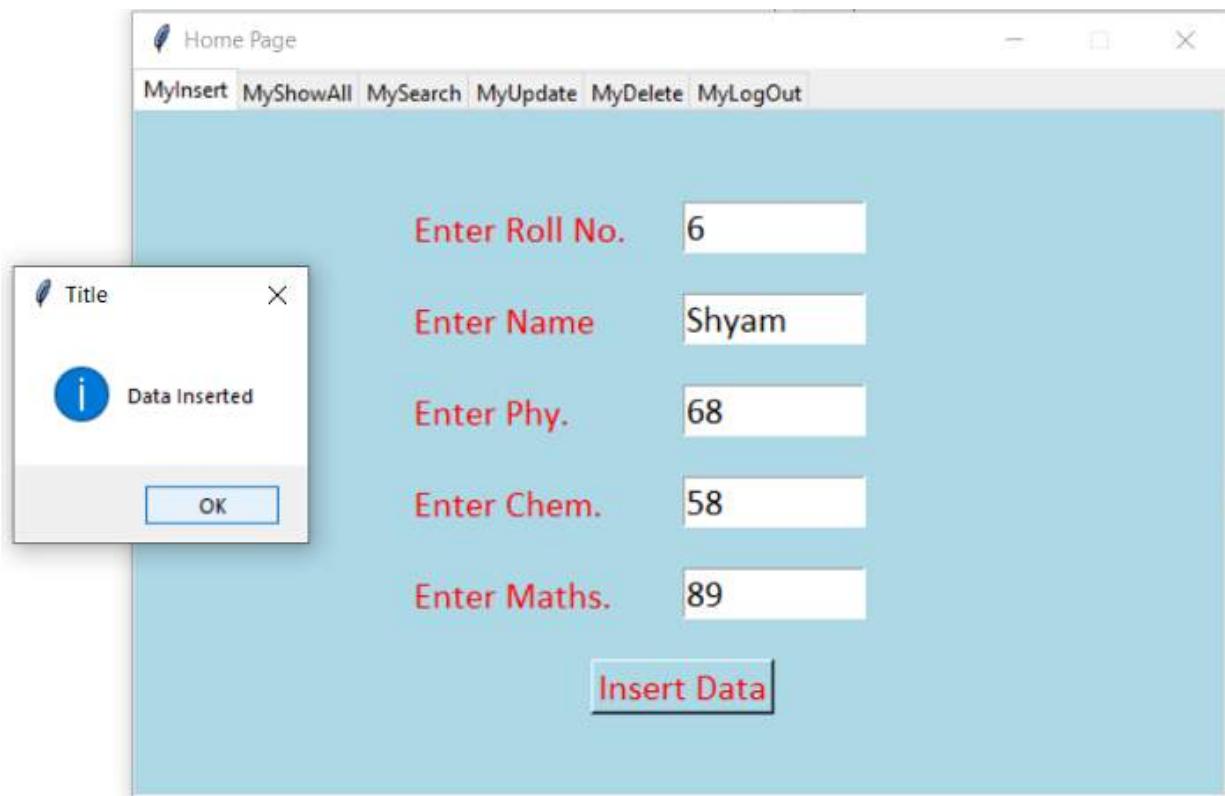


Figure 11.6: Inserting the data

8. In **Figure 11.7**, we can see that the database is updated:

Database Structure					Browse Data	Edit Pragmas	Execute SQL
Table: ins							
URNO	UNAME		UPHY	UCHE	UMATHS		
Filter	Filter	Filter	Filter	Filter	Filter	Filter	
1 1	Ram		88	90	100		
2 3	Nilesh		89	92	99		
3 4	Saurabh		92	100	100		
4 5	Divya		100	90	92		
5 6	Shyam		68	58	89		

Figure 11.7: Database is updated

9. On clicking the **MyShowAll** tab, the following form shown in *Figure 11.8* is displayed:



Roll No.	Name	Phy.	Chem.	Maths
1	Ram	88	90	100
3	Nilesh	89	92	99
4	Saurabh	92	100	100
5	Divya	100	90	92
6	Shyam	68	58	89

Figure 11.8: Form displayed on focusing MyShowAll tab

10. On clicking the **MySearch** tab, we are entering the **Roll No.** to see the data, as shown in *Figure 11.9*:

Home Page

MyInsert MyShowAll MySearch MyUpdate MyDelete MyLogOut

Roll No. Search

Name is: Nilesh

Phy : 89

Chem : 92

Maths : 99

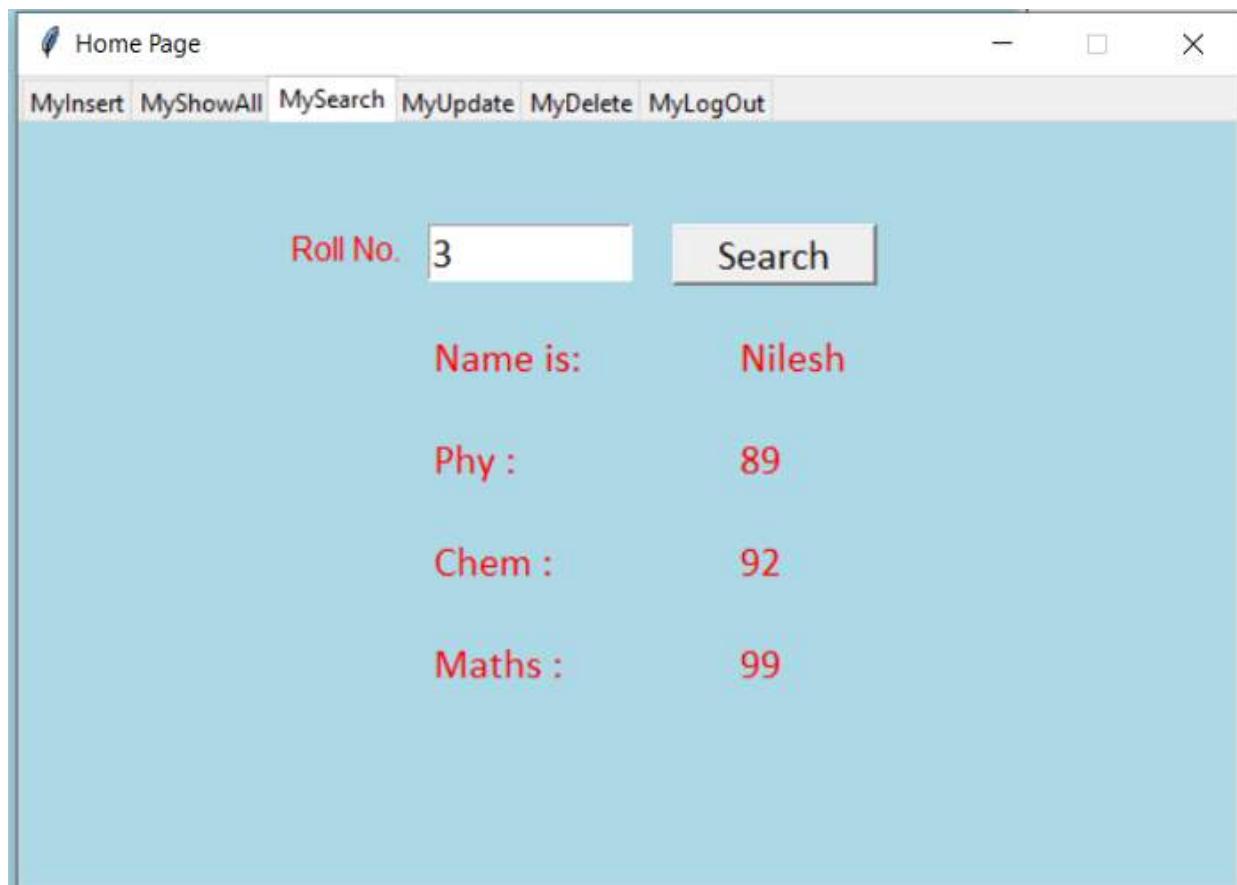


Figure 11.9: Entering Roll No. to see the data on focusing MySearch tab

11. On clicking the **MyUpdate** tab, the following form shown in [Figure 11.10](#) is displayed:

Home Page

MyInsert MyShowAll MySearch MyUpdate MyDelete MyLogOut

Roll No. Retrieve

Name is:

Phy :

Chem :

Maths :

Update

Figure 11.10: Form displayed on focusing MyUpdate tab

12. On clicking the **Update** button, the database data is updated as shown in [Figure 11.11](#):

Database Structure Browse Data Edit Pragmas Execute SQL

Table: ins

	URNO	UNAME	UPHY	UCHE	UMATHS
	Filter	Filter	Filter	Filter	Filter
1	1	Ram	88	90	100
2	3	Nilesh	89	92	99
3	4	Saurabh	92	100	100
4	5	Divya	100	90	92
5	6	Shyam	68	58	90

Figure 11.11: Updated database

13. On clicking the **MyDelete** tab, the following form shown in [Figure 11.12](#) is displayed:

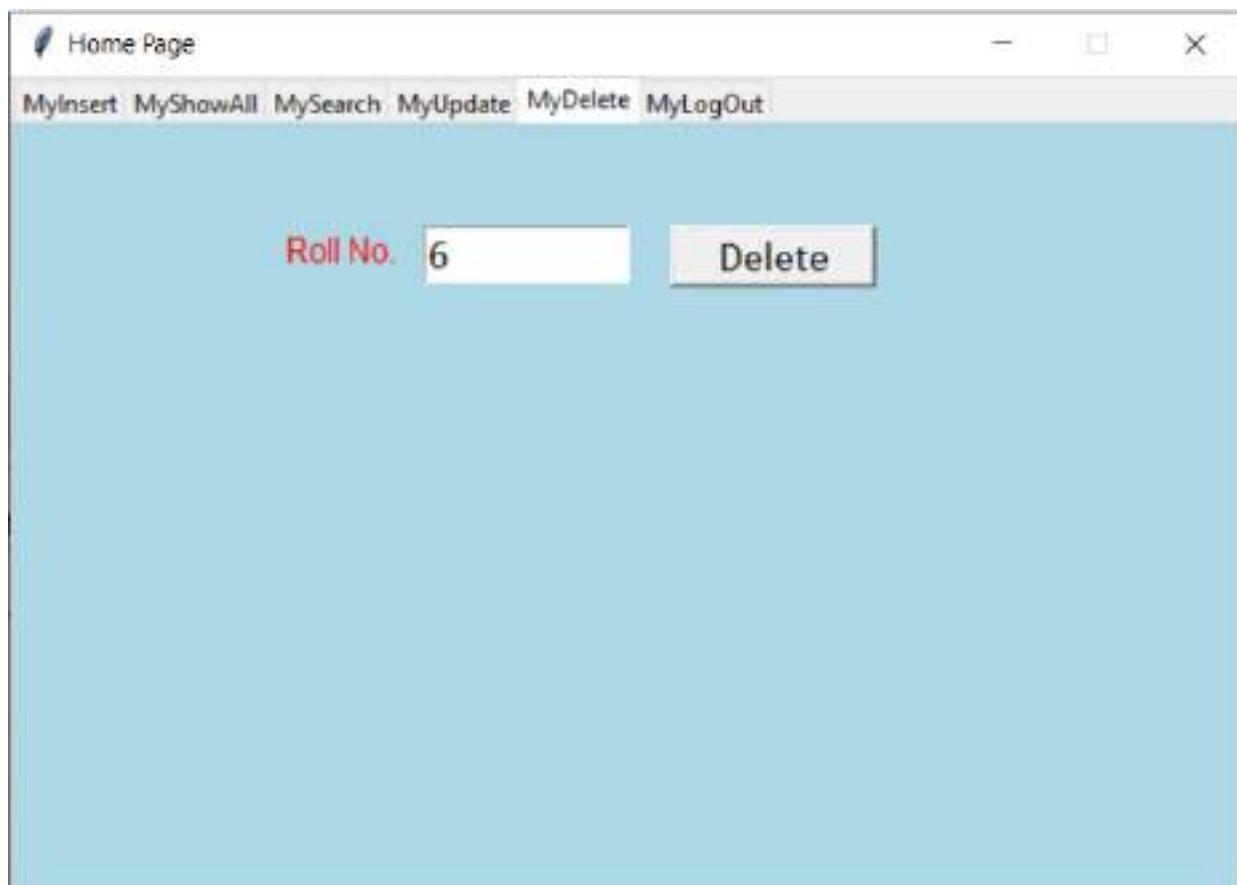


Figure 11.12: Form displayed on focusing MyDelete tab

14. User can delete the data by clicking on the **Delete** button, as shown in [Figure 11.13](#):

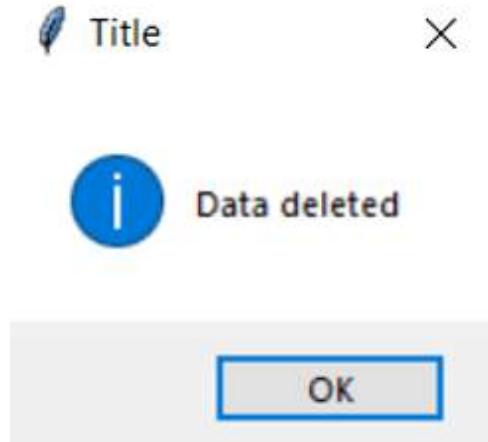


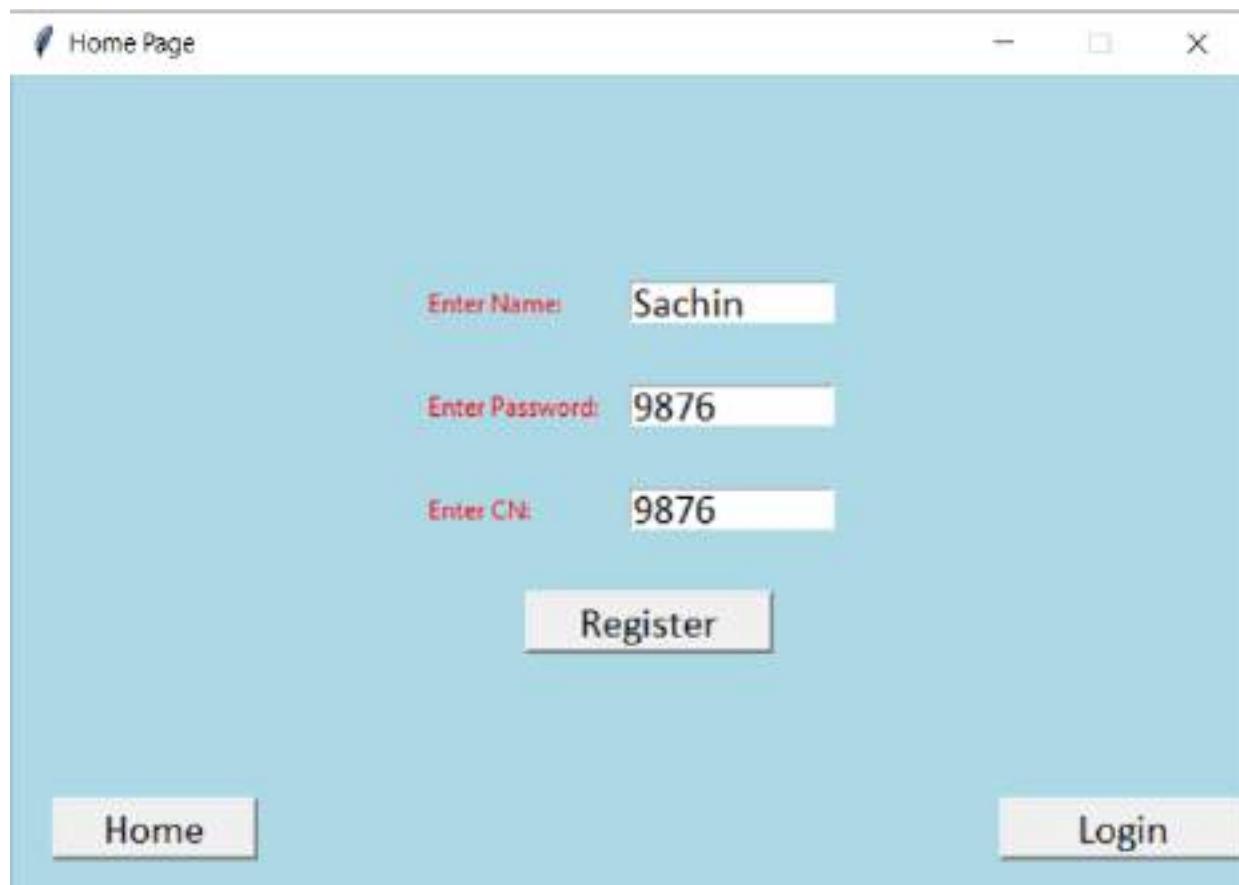
Figure 11.13: Delete button

15. The database now looks as shown in [Figure 11.14](#):

	URNO	UNAME	UPHY	UCHE	UMATHS
1	1	Ram	88	90	100
2	3	Nilesh	89	92	99
3	4	Saurabh	92	100	100
4	5	Divya	100	90	92

Figure 11.14: Updated database

16. On clicking the **MyLogOut** tab, the user will navigate to the home page consisting of **Login** and **Register** button.
17. On clicking the **Register** button, the user will navigate to the following form shown in [Figure 11.15](#), where user can register for login:



A screenshot of a registration form window titled "Home Page". The window contains three input fields: "Enter Name:" with the value "Sachin", "Enter Password:" with the value "9876", and "Enter CN:" with the value "9876". Below these fields is a "Register" button. At the bottom left is a "Home" button, and at the bottom right is a "Login" button.

Enter Name:	Sachin
Enter Password:	9876
Enter CN:	9876

Register

Home Login

Figure 11.15: Form to register for login

18. On clicking the **Register** button, a new user is registered into the database where the user can login, as shown in *Figure 11.16*:

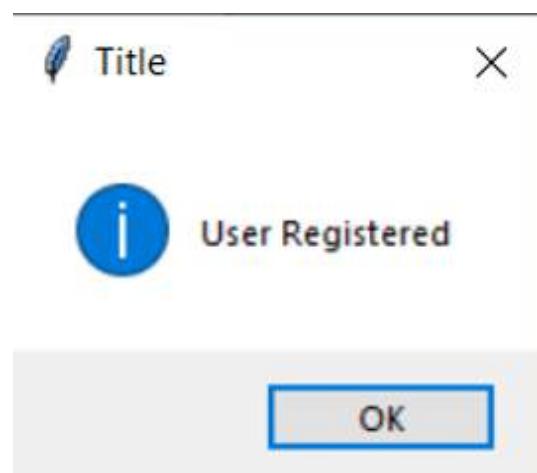
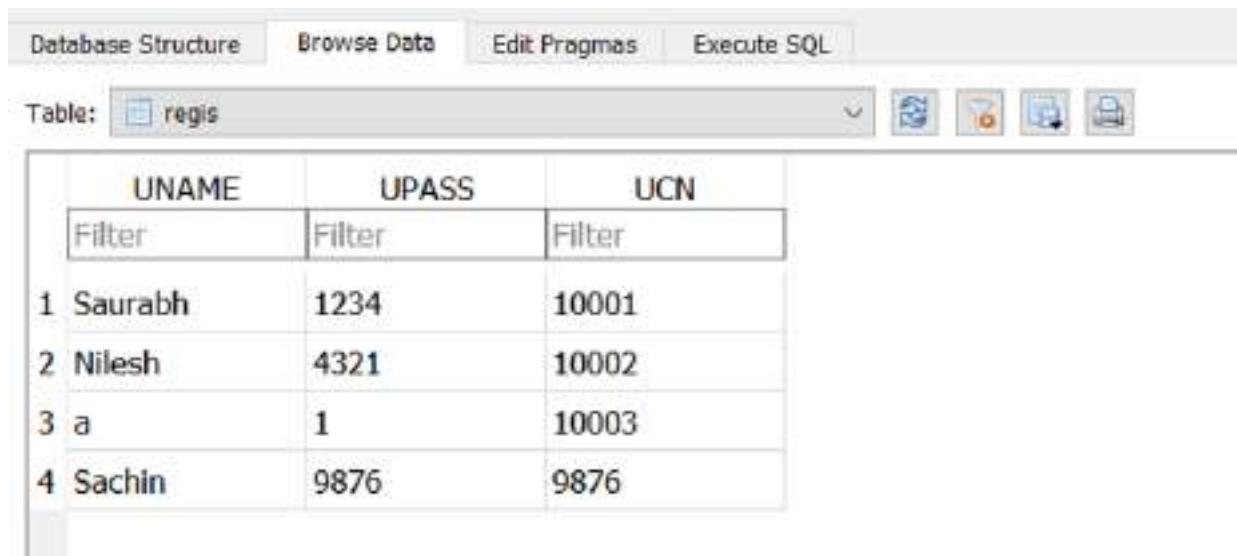


Figure 11.16: User registered

19. The updated database looks as shown in *Figure 11.17*:



The screenshot shows a SQLite database browser interface. At the top, there are tabs: 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. Below the tabs, it says 'Table: regis'. There are several icons for database operations like insert, update, delete, and export. The main area displays a table with three columns: 'UNAME', 'UPASS', and 'UCN'. The table has four rows of data:

	UNAME	UPASS	UCN
1	Saurabh	1234	10001
2	Nilesh	4321	10002
3	a	1	10003
4	Sachin	9876	9876

Figure 11.17: Updated database after registration

20. On clicking the **Login** page, a new user can login.

Conclusion

In this chapter, we learned how to connect any GUI program with sqlite3 database by creating a small GUI application using tkinter library. Different steps of interacting with sqlite3 database were explored from initially importing the module, then connecting to the database, creating a cursor object followed by execute queries and committing the changes and finally closing the connection. We saw step by step approach of interacting our GUI application with sqlite3 database by creating Login form, Registration form, Home Page, various tabs as per need and so on.

Points to remember

- **Import the required libraries:** To connect to the SQLite3 database, we need to import the sqlite3 library in addition to the tkinter library.
- **Establishing a database connection:** To connect to the sqlite3 database, use the `sqlite3.connect()` function. The parameter for this

function is the name of the database file.

- **Creating a cursor:** After establishing a connection, use the connection object's **cursor()** method to generate a cursor. Executing SQL commands and obtaining data from the database are done using this cursor.
- **Make GUI components:** Create a GUI that communicates with the database using tkinter widgets like Label, Entry, Button, and so on.
- Use cursor object to execute queries. To commit changes to the database, use the **commit()** method.
- **Close the database connection:** To close the database connection, use the connection object's **close()** method.
- **Test your programme:** Make sure to thoroughly test your application to make sure it functions as expected and gracefully handles errors.

Questions

1. Explain the steps to connect a GUI application to the sqlite3 database.
2. Create a Python code such that a GUI application created using tkinter library can interact with a sqlite3 database. Then try connecting your GUI applications with any other database such as mysql.

Index

A

anchors [19](#), [20](#)
 constants [20-23](#)
application-level binding [62](#)
askdirectory function [285](#)
askokcancel() [198](#), [199](#)
askopenfile function [286](#), [287](#)
askopenfilename function [288](#)
askopenfilenames function [289](#), [290](#)
askquestion() [197](#), [198](#)
askretrycancel() [200](#), [201](#)
asksaveasfile function [291](#), [292](#)
asksaveasfilename function [293](#)
askyesno() [199](#), [200](#)

B

basic Python GUI program [4-7](#)
bitmaps [25](#), [26](#)
BooleanVar() [42](#), [43](#)

C

classes
 for GUI creation [47-50](#)
colors
 activebackground [11](#)
 activeforeground [12](#)
 background [11](#), [12](#)
 disabledforeground [13](#), [14](#)
 foreground [12](#), [13](#)
 highlightbackground [14](#), [15](#)
 selectbackground [15](#)
 selectforeground [15](#), [16](#)
cursors [26](#), [27](#)

D

dimensions [8](#)
 borderwidth [8](#)
 height [10](#)

highlightthickness [8, 9](#)
padX [9](#)
padY [9](#)
underline [10](#)
width [11](#)
wraplength [10](#)
DoubleVar() [45-47](#)

E

event types, tkinter
 ButtonPress or Button [60](#)
 ButtonRelease [61](#)
 Configure [61](#)
 Enter [61](#)
 Event details [61](#)
 FocusIn [61](#)
 FocusOut [61](#)
 keyboard event [61](#)
 Keypress or Key [61](#)
 KeyRelease [61](#)
 Leave [61](#)
 Motion [61](#)
 mouse event [61](#)
 Mousewheel [61](#)

F

file selection
 handling, in tkinter [284-299](#)
fonts [16](#)
 font object, creating [16, 17](#)
 tuple, using [19](#)

G

Graphical User Interface (GUI) [1](#)
 element [53](#)
GUI application
 displaying [314-339](#)
GUI creation
 advantages [50](#)
 classes and objects, using [47-50](#)
GUI interaction
 using, with sqlite3 database [314](#)

I

inbuilt variable classes 39, 40

BooleanVar() 42, 43

DoubleVar() 45-47

IntVar() 44, 45

StringVar() 40-42

instance-level binding 62

IntVar() 44, 45

K

keyboard events 62

L

lambda expressions

advantages 69

M

mouse events 62

mycall(event) method 64

mycallme(event) method 64

O

objects

for GUI creation 47-50

P

Python tkinter geometry management 28

geometry method 35, 36

grid() 29-32

pack() 28, 29

place() 32-34

Python tkinter GUI

standard attributes 8

R

relief styles 24

S

showerror() 196

showinfo() 194, 195

showwarning() 195

sqlite3 browse data 331

sqlite3 database

GUI interaction, with [314](#)
sqlite3 database structure [331](#)
standard attributes, Python tkinter GUI
anchors [19](#)
colors [11](#)
dimensions [8](#)
fonts [16](#)
StringVar() [40-42](#)

T

Tcl/Tk [2](#)
tkinter [2, 3](#)
 file selection, handling [284-299](#)
tkinter Button Widget [54-59](#)
 command option [54](#)
 event bindings [60](#)
 events [60, 61](#)
 examples [63-80](#)
 justify option [54](#)
 keyboard events [62](#)
 mouse events [62](#)
tkinter Canvas widget [267](#)
 examples [268-281](#)
 options [268](#)
tkinter Checkbutton widget [81-88](#)
 command option [81](#)
 deselect method [89](#)
 example [89, 90](#)
 flash method [89](#)
 invoke method [89](#)
 offvalue option [81](#)
 onvalue option [81](#)
 options [81](#)
 select method [88](#)
 text option [81](#)
 toggle method [89](#)
 variable option [81](#)
tkinter Combobox widget [164](#)
 example [165-172](#)
 methods [165](#)
 options [165](#)
tkinter Entry widget [106-120](#)
 command option [106](#)
 delete () method [110](#)
 exportselection option [107](#)

get() method 111
icursor(index) method 111
index(index) method 113
insert(index,mystr) method 112
option 106
select_adjust(index) method 113
selectborderwidth option 107
select_clear() method 115
select_from(index) method 115
select_present() method 115
select_range(start, end) method 115
select_to(index) method 115
show option 107
state option 107
textvariable option 107
validatecommand option 121
validate option 121
validation 121-124
xscrollcommand option 107
xview(index) method 120
xview_scroll(number, what) method 120

tkinter Frame Widget 204-208
tkinter LabelFrame widget 209
examples 209-212
options 209

tkinter Label widget 176
creating 177-190
options 176

tkinter Listbox widget 236
examples 237-245
methods 236, 237
options 236

tkinter Menubutton widget 263
examples 264-267
options 264

tkinter Menu widget 248
examples 249-263
methods 249
options 248

tkinter MessageBox widget 193, 194
askokcancel() 198, 199
askquestion() 197, 198
askretrycancel() 200, 201
askyesno() 199, 200
showerror() 196
showinfo() 194, 195

showwarning() 195
tkinter Message widget 191
 options 191-193
tkinter OptionMenu widget 100-102
tkinter PanedWindow widget 214, 215
 examples 216-220
 methods 215, 216
 options 215
tkinter Radiobutton widget 91
 command option 91
 deselect method 98
 examples 92-100
 flash method 98
 image option 91
 indicatoron option 92
 invoke method 98
 options 91, 92
 selectcolor option 91
 selectimage option 91
 select method 98
 state option 91
 text option 91
 textvariable option 91
 value option 91
 variable option 92
tkinter Scale widget 138
 examples 140-145
 methods 139
 options 139
tkinter Scrollbar widget 126
 command option 126
 elementborderwidth option 126
 get method 127
 jump option 126
 methods, using 127
 orient option 126
 pack method 127
 repeatdelay option 126
 repeatinterval option 126
 scrollbar, attached to Canvas 130, 131
 scrollbar, attached to Entry 131, 132
 scrollbar, attached to Listbox 127, 128
 scrollbar, attached to Text 128, 130
 set(first, last) method 127
 takefocus option 126
 troughcolor option 127

tkinter Spinbox widget [132](#)
 command option [133](#)
 delete(startindex [,endindex]) method [134](#)
 examples [134-138](#)
 format option [133](#)
 from_option [133](#)
 get(startindex [,endindex]) method [134](#)
 identify(x,y) method [134](#)
 index(index) method [134](#)
 insert(index [,string]...) method [134](#)
 option [133](#)
 repeatdelay option [133](#)
 repeatinterval option [133](#)
 selection_clear() method [134](#)
 selection_get() method [134](#)
 textvariable option [133](#)
 to option [133](#)
 validatecommand option [133](#)
 validate option [133](#)
 wrap option [133](#)
 xscrollcommand option [133](#)

tkinter Tabbed/Notebook widget [212](#)
 examples [213, 214](#)
 options [213](#)

tkinter Text widget [145](#)
 examples [147-164](#)
 methods [146, 147](#)
 options [145, 146](#)

tkinter Toplevel widget [221](#)
 example [222-232](#)
 methods [221, 222](#)
 options [221](#)

Tool Command Language (Tcl) [2](#)
trace, in tkinter [308](#)
 trace_add() [309](#)
 trace_info() [309-311](#)
 trace_remove() [309](#)

W

widget information
 obtaining [302-308](#)