APPENDIX B GUI Programming With tkinter Package

B.1 INTRODUCTION

Python is a language which can be used across different applications including GUI programs. Although there are many ways in which you can create a GUI application, we will discuss the `tkinter` package.

`tkinter` is distributed along with Python software. It is a cross-platform, stable, reliable, and easy to learn package. It has a variety of commonly used GUI elements (like buttons, menus, labels, entry areas, etc.) which can be used to build the interface. These elements are called *widgets*.

B.2 WIDGETS

Widgets are the building blocks of GUI programming and are used to display information or get input from the user. Some frequently used widgets are given in Table B.1.

Table B.1 Widgets and their Functions

Widget	Function
`Frame`	Used as a container to house other widgets and add borders.
`Label`	To display text or images.
`Labelframe`	By default, this frame displays a border and title.
`Button`	Used to call a function when clicked.
`Checkbutton`	Used to create checkbox for toggling a value.
`Radiobutton`	Used to create standard radio buttons.
`Entry`	Used for single line text entry which can be edited.
`Text`	Used for multiple line text entry which can be edited.
`Message`	Used for multiple line display text that can be styled.
`Combobox`	To do a single line text entry with a drop-down select list.
`Listbox`	To create multiple line select list.
`Scrollbar`	To create vertical or horizontal scroll.
`Scale`	To create a sliding scale that can be clicked and dragged with mouse.
`Canvas`	To draw lines, circles, arcs, ovals, and rectangles.
`Menu`	To create menu bar for the top of the window, which includes parameters like `"File"`, `"Edit"`, and `"View"`.

In the small GUI application that we will be writing now, a *tree* of widgets will be created such that each widget will have a parent widget, all the way up to the *root window* of our application. For example, a button or a text field will be added in the containing window.

The widget classes has a good collection of methods for configuring the GUI's appearance. For example, setting the location, content, color, layout, etc. of widget and handling various kinds of user-driven events. Once we have constructed the backbone of our GUI, we will need to customize it by integrating it with our internal application class.

Note Window manager which handles windows is a part of the operating system. So, during GUI programming we need not write the codes for closing the window.

Try executing Example B.1. You should be able to see a window with a title, a text label, and two buttons—one which prints a message in the console, and one which closes the window. The window should have all the normal properties of any other window you encounter in your window manager—you are probably able to drag it around by the title bar, resize it by dragging the frame, and maximize, minimize, or close it using buttons on the title bar.

Example B.1 Program to print the version of `tkinter`

```
import tkinter
print(tkinter.TkVersion)     #prints the version of tkinter module
```

OUTPUT

```
8.5
```

Example B.2 Program to display the `tkinter` window

```
from tkinter import Tk
root = Tk()
root.mainloop()
```

OUTPUT

Note that `Tk` is the class which is used to create the *root* window (main window) of the application. Although you can create multiple windows in your application, there should be only one root in the application.

Example B.3 Program to set the title of the Tkinter window

```
from tkinter import Tk
root = Tk()
root.title("Fun with Python")
root.mainloop()
```

OUTPUT

In the aforementioned program, we have not used inheritance to utilize `tkinter` package rather, we have used *composition* to associate our tree of widgets with our class. However, inheritance could have also been used to extend one of the widgets in the tree with our custom functions.

`root.mainloop()` is a method on the main window which is executed when we run our application. The method loops forever, waiting for events from the user, until the user exits the program either by closing the window, or by terminating the program with a keyboard interrupt in the console.

Example B.4 Program to print the screen size

```
from tkinter import Tk
root = Tk()
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
print("Screen width:", screen_width)
print("Screen height:", screen_height)
```

OUTPUT

```
('Screen width:', 1366)
('Screen height:', 768)
```

Example B.5 Program to make the window fullscreen

```
from tkinter import Tk
root = Tk()
root.attributes('-fullscreen', True)
root.mainloop()
```

Example B.6 Program to resize the window at specified location

```
from tkinter import Tk
root = Tk()
# Make window 400x200 and place at position (150,150)
root.geometry("400x200+150+150")
root.mainloop()
```

Example B.7 Program to create a label as a child of the root window

```
from tkinter import Tk, Label
root = Tk()
msg = Label(root, text='Welcome to
programming with Tkinter!')
msg.pack()
root.mainloop()
```

OUTPUT

Note The pack() method is used on each widget to position it inside its parent.

Example B.8 Program to change the font of the text

```
from tkinter import Tk, Message
root = Tk()
msg = Message(root, text='Welcome to
programming with Tkinter!')
msg.config(font=('callibri', 24, 'italic
bold underline'))
msg.pack()
root.mainloop()
```

OUTPUT

Example B.9 Program to display two labels with different colored background

```
from tkinter import Tk, Label, Y, RIGHT
root = Tk()
label1 = Label(root, text='Welcome To Python!', background='red')
label2 = Label(root, text='Have a Good Day', background='green')
# Some of the packing options:
# - fill: widget expands to take up any extra space (X, Y, or BOTH)
# - padx/pady: outter padding
# - ipadx/ipady: inner padding
# - side: which side to stack from. Default is TOP (to bottom)
label1.pack(fill=Y, padx=10, ipady=25, side=RIGHT)  # Pack from right to left
label2.pack(fill=Y, padx=20, ipady=40, side=RIGHT)
root.mainloop()
```

OUTPUT

Note Label is a static element which does not do anything by default except displaying some information.

Example B.10 Program to display an image

```
import sys
from tkinter import Tk, Label, PhotoImage
root = Tk()
img = PhotoImage(file=sys.argv[1])
IMG = Label(root, image=img)
IMG.pack()
root.mainloop()
```

OUTPUT

```
The image will be displayed in the window
```

Example B.11 Program to display two buttons and print a message when a button is clicked

```
from tkinter import Tk, Button
root = Tk()
# Exit window will close the GUI window when clicked
exitButton = Button(root, text='Exit Program', command=root.destroy)
exitButton.pack()
# To write a message on the screen and not on GUI window
def my_callback():
    print("You clicked the Message Button....")
msg_button = Button(root, text='Click Here', command=my_callback)
msg_button.pack()
root.mainloop()
```

OUTPUT

Example B.12 Program to print a colored text on a colored background of GUI window

```
from tkinter import Tk, Label
root = Tk()
label = Label(root, text='Welcome to GUI programming in Python !!!')
label.pack()
```

```
label.config(foreground='yellow', background='blue', text='Updated text!')
root.mainloop()
```

OUTPUT

Example B.13 Program to display a menu on the menu bar

```
from tkinter import Tk, Menu
root = Tk()
# Create main menu bar
menu_bar = Menu(root)
# Create the submenu, keeo tearoff denotes that menu can pop out
fileMenu = Menu(menu_bar, tearoff=0)
# Add commands to submenu
fileMenu.add_command(label="Stop", command=root.destroy)
fileMenu.add_command(label="Kill", command=root.destroy)
fileMenu.add_command(label="Exit", command=root.destroy)
# Add the File drop down sub-menu in the main menu bar
menu_bar.add_cascade(label="File", menu=fileMenu)
root.config(menu=menu_bar)
root.mainloop()
```

OUTPUT

Example B.14 Program that has an entry box in which the message entered by user is printed on the IDLE screen

```
from tkinter import Tk, Entry, Button, INSERT
root = Tk()
# Create an entry box
entry = Entry(root)
entry.pack()
# Print the contents of entry box in console
```

```
def printMsg():
    print(entry.get())
# Create a button that when clicked will print the contents of the entry box
button = Button(root, text='Print content', command=printMsg)
button.pack()
root.mainloop()
```

OUTPUT

Note Buttons are designed to cause something to happen when they are clicked. The command keyword parameter is used to specify the function which handles each button's click event.

Example B.15 Program to display the text on the console when a label is pressed

```
from tkinter import Tk, Label
root = Tk()
label = Label(root, text='Message printing label.....')
label.pack()
def my_callback():
    print('Welcome to GUI Programming of Python')
# Bind left mouse button click on label
label.bind("<Button-1>", lambda e:my_callback())
root.mainloop()
```

OUTPUT

Example B.16 Program to display a pop-up dialog box

```
from tkinter import messagebox
title = 'Customer Feedback'
text = 'Did you like our Customer Service?'
reply = messagebox.askquestion(title, text)
if reply == 'yes':
    print('Thank you very much... ')
else:
```

```
print('We regret the inconvenience. Please give us another chance.')
```

OUTPUT

Example B.17 Program to create a top-level window which will be closed when the button is clicked

```
from tkinter import Tk, Toplevel, Button
root = Tk()
# Create new top level window
top_level_window = Toplevel()
top_level_window.title('Top Level Window')
# Destroy Top Level Window window
def destroy_top_level_window():
    top_level_window.destroy()
closeButton = Button(root, text='Close Top Level Window', command=destroy_top_
level_window)
closeButton.pack()
root.mainloop()
```

OUTPUT

Example B.18 Program to put focus on top-level window

```
from tkinter import Tk, Toplevel
root = Tk()
top_level_window = Toplevel()
```

```
top_level_window.title('New Top Level Window')
# Focus on window
top_level_window.focus_force()
root.mainloop()
```

OUTPUT

Example B.19 Program to draw colored shapes on canvas

```
from tkinter import Tk, Canvas
root = Tk()
canvas = Canvas(root, width=250, height=200)
canvas.pack()
# Draw orange dashed line
canvas.create_line(0, 0, 250, 200, fill='orange', dash=(5, 15))
# Draw yellow rectangle at (100,50) to (150,60)
canvas.create_rectangle(100, 50, 150, 60, fill='yellow')
# Draw oval(circle) from (30,30) to (50,50)
canvas.create_oval(30, 30, 50, 50, fill='green')
root.mainloop()
```

OUTPUT