UNIT-5 (Tkinter GUI, Event Driven Programming, Widgets)

The Behavior of Terminal-Based Programs and GUI-Based Programs, Label, Entry and Button widget; Tkinter Geometry methods (pack(), grid(), place()); Event-Driven Programming, Command Buttons and Responding to Events; CheckButton and Radiobutton widgets;
Menu and Menu button widgets; Listbox and Scrollbar widgets; Messagebox and Toplevel widget; File Dialog widget;

5.0 Differences Between Terminal-Based Programs and GUI-Based Programs:

Aspect	Terminal-Based Programs	GUI-Based Programs
User Interaction	- User interacts via text-based commands.	- Users interact using graphical elements like buttons, menus.
	- Requires knowledge of specific commands or syntax.	- More intuitive, requiring less technical knowledge.
Ease of Use	- Steeper learning curve for non-technical users.	- User-friendly with visual cues and interactions.
Design	- Linear, text-driven interface with limited customization.	- Highly customizable with layouts, colors, and interactive widgets.
Efficiency	- Lightweight and resource-efficient.	- May require more system resources due to graphical components.
Use Cases	- Ideal for developers, system admins, scripting tasks.	- Suitable for applications involving broader audiences (e.g., desktop apps).
Examples	- Command-line tools like Git, SSH, Python CLI apps.	- Applications like browsers, games, and data-entry tools.

5.0.1 Introduction to GUI:

A GUI (Graphical User Interface) allows users to interact with electronic devices using graphical icons and visual indicators, rather than text-based interfaces, typewritten command labels, or text navigation. Tkinter is Python's standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk.

Tkinter is a combination of “Tk” and “inter”, where Tk stands for Tool kit and inter stands for interface i.e., Tkinter is a python interface to the Tk tool kit.

A popular GUI Tool kit that was originally developed for Tcl programming language. Tk provides a rich collection of widgets for building interactive applications.

1. **Buttons, Labels, Entry fields:** For basic interactions.
2. **Checkboxes, Radio Buttons:** For user selections.
3. **Canvas:** For drawing shapes or graphics.
4. **Menus and Toolbars:** For navigation and advanced functionality.

5.0.2 Tkinter's Role in GUI Development

Tkinter is Python's standard library for creating GUI-based programs. It provides a toolkit for building graphical interfaces with minimal effort. Here's how it facilitates GUI development:

- **Ease of Use:** Tkinter's simple syntax allows developers—even beginners—to create windows, menus, and widgets like buttons and entry fields quickly.
- **Widget Variety:** It offers a wide range of widgets to design diverse interfaces, such as labels, text boxes, frames, canvases, progress bars, and more.
- **Event Handling:** Tkinter includes mechanisms to handle user actions (events), such as button clicks or text input, enabling dynamic and interactive programs.
- **Cross-Platform Compatibility:** Since Tkinter is part of Python's standard library, it works seamlessly across different operating systems.
- **Customization:** Developers can enhance the aesthetics by modifying widget properties, setting colors, fonts, layouts, etc.

5.0.3 Steps to create a simple GUI application using Tkinter:

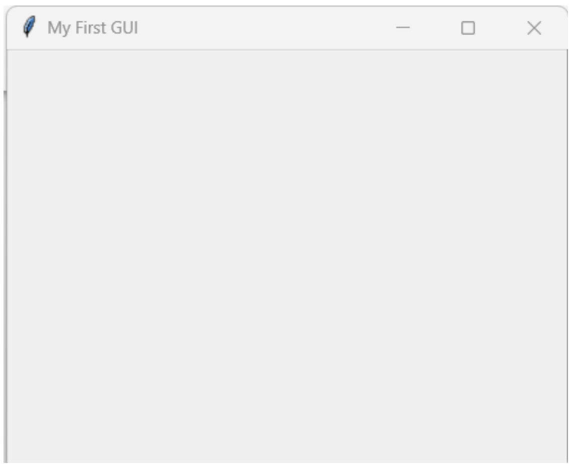
1. import tkinter module
2. Create the main application window
3. Add the widgets like labels, buttons, frames etc., to the window.
4. Call the main event loop so that the actions can take place on the users computer screen.

5.0.3 Simple GUI Creation with Methods title() and geometry():

Tkinter provides a Tk class to create a window. The title() method sets the title of the window, and geometry() sets the size and placement of the window.

Example:

```python
from tkinter import Tk

# Create the main window
root = Tk()

# Set the title of the window
root.title("My First GUI")

# Set the size and position of the window
root.geometry("400x300")

# Start the GUI event loop
root.mainloop()
```

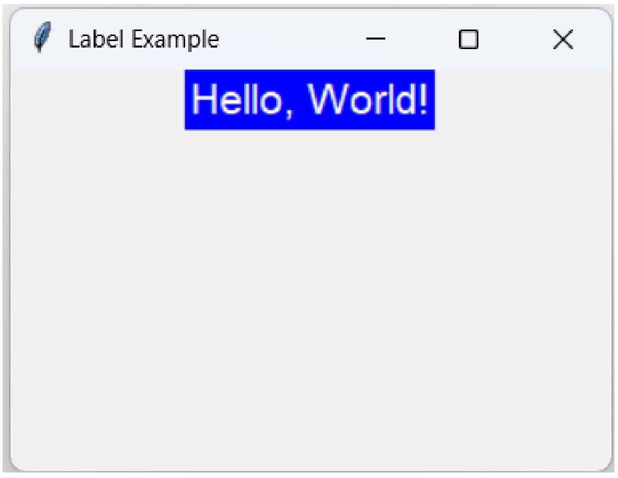


5.1 Program on Label Creation:

A Label widget is used to display text or images. You can customize various options such as text, font, background color (bg), and foreground color (fg).

Example:

```
from tkinter import Tk, Label

# Create the main window
root = Tk()
root.title("Label Example")
root.geometry("300x200")

# Create a label with text, font, background, and foreground colors
label = Label(root, text="Hello, World!", font=("Arial", 16), bg="blue", fg="white")
label.pack()

# Start the GUI event loop
root.mainloop()
```



The pack() method is one of the three geometry managers in Tkinter (the other two being grid() and place()). It organizes widgets in blocks before placing them in the parent widget.

Default Behavior: By default, pack() will add the widget to the window from top to bottom, centering it horizontally.

The root.mainloop() is a method in Tkinter that is used to run the main loop of your application. It's an essential part of any Tkinter-based GUI application.

- **Main Loop:** The main loop is a continuous loop that runs in the background of your application. It waits for events (such as button clicks, key presses, mouse movements, etc.) and dispatches them to the appropriate event handlers.
- **Blocking Call:** root.mainloop() is a blocking call, which means it will run indefinitely until the main window is closed. This keeps your application responsive and able to react to user input.

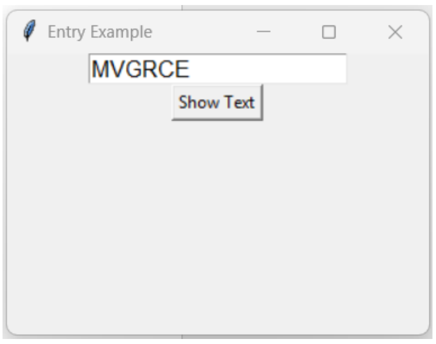
5.2 Program on Entry Creation:

An Entry widget is used to accept a single line of text input from the user. You can customize text color (fg), background color (bg), width, font, and use methods like get(), delete(), and insert().

Program:

```
from tkinter import Tk, Entry, Button

# Function to show text from entry widget
def show_text():
    entered_text = entry.get()
    print(entered_text)

# Create the main window
root = Tk()
root.title("Entry Example")
root.geometry("300x200")

# Create an entry widget with text color, background color, width, and font
entry = Entry(root, fg="black", bg="white", width=20, font=("Arial", 12))
entry.pack()

# Create a button to show the entered text
button = Button(root, text="Show Text", command=show_text)
button.pack()

# Start the GUI event loop
root.mainloop()
```

```
MVGRCE
```



Explanation:

The statement from tkinter import Tk, Entry, Button is an import statement in Python that allows you to access specific classes from the Tkinter module. Let's break it down:

- **from tkinter import:** This part tells Python to import specific items from the Tkinter module.

- **Tk:** This is the Tk class from the Tkinter module. It is used to create the main window of your application. When you create an instance of the Tk class, it initializes the Tkinter application and creates the main window.
- **Entry:** This is the Entry class from the Tkinter module. It is used to create a single-line text entry field within the Tkinter window. Entry widgets are commonly used to take input from the user.
- **Button:** This is the Button class from the Tkinter module. It is used to create clickable buttons within the Tkinter window. Buttons can be configured to perform specific actions when clicked.

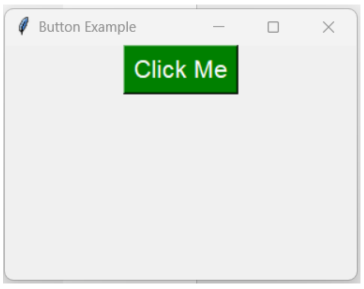
5.3 Program on Button Creation:

A **Button** widget is used to execute a command when clicked. You can customize the text, font, background color (bg), foreground color (fg), active background color (activebackground), and active foreground color (activeforeground).

Program:

```
from tkinter import Tk, Button

# Function to handle button click
def on_click():
    print("Button clicked!")

# Create the main window
root = Tk()
root.title("Button Example")
root.geometry("300x200")

# Create a button with text, font, colors, and a command
button = Button(root, text="Click Me", font=("Arial", 16), bg="green", fg="white",
                activebackground="yellow", activeforeground="red", command=on_click)
button.pack()

# Start the GUI event loop
root.mainloop()
```

```
Button clicked!
```



Explanation: The statement from tkinter import Tk, Label, Button is a Python import statement that allows you to access specific classes from the Tkinter module.

- **from tkinter import:** This part of the statement tells Python to import specific items from the Tkinter module.
- **Tk:** This is the Tk class from the Tkinter module. It is used to create the main window of your application. When you create an instance of the Tk class, it initializes the Tkinter application and creates the main window.
- **Label:** This is the Label class from the Tkinter module. It is used to create text or image labels within the Tkinter window. Labels are often used to display static text or images.
- **Button:** This is the Button class from the Tkinter module. It is used to create clickable buttons within the Tkinter window. Buttons can be configured to perform specific actions when clicked.

5.4 Introduction to Tkinter Geometry Methods:

Tkinter provides three geometry managers to organize widgets within a parent widget:

1. **pack():** Arranges widgets in blocks, filling the available space.
2. **grid():** Organizes widgets in a table-like structure with rows and columns.
3. **place():** Allows precise positioning of widgets using coordinates.

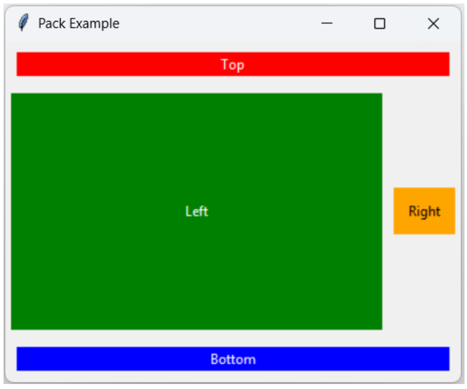
5.4.1 Programs on pack() with Options:

The pack() method can be customized using various options:

- **expand**: If set to True, the widget expands to fill any extra space.
- **side**: Specifies which side of the parent widget to pack against (TOP, BOTTOM, LEFT, RIGHT).
- **fill**: Determines if the widget should expand to fill extra space (NONE, X, Y, BOTH).
- **ipadx, ipady**: Adds internal padding inside the widget (horizontal and vertical).
- **padx, pady**: Adds external padding outside the widget (horizontal and vertical).

Program:

```
from tkinter import Tk, Label

root = Tk()
root.title("Pack Example")
root.geometry("400x300")

label1 = Label(root, text="Top", bg="red", fg="white")
label1.pack(side="top", fill="x", padx=10, pady=10)

label2 = Label(root, text="Bottom", bg="blue", fg="white")
label2.pack(side="bottom", fill="x", padx=10, pady=10)

label3 = Label(root, text="Left", bg="green", fg="white")
label3.pack(side="left", expand=True, fill="both", padx=5, pady=5)

label4 = Label(root, text="Right", bg="orange", fg="black")
label4.pack(side="right", ipadx=10, ipady=10, padx=5, pady=5)

root.mainloop()
```



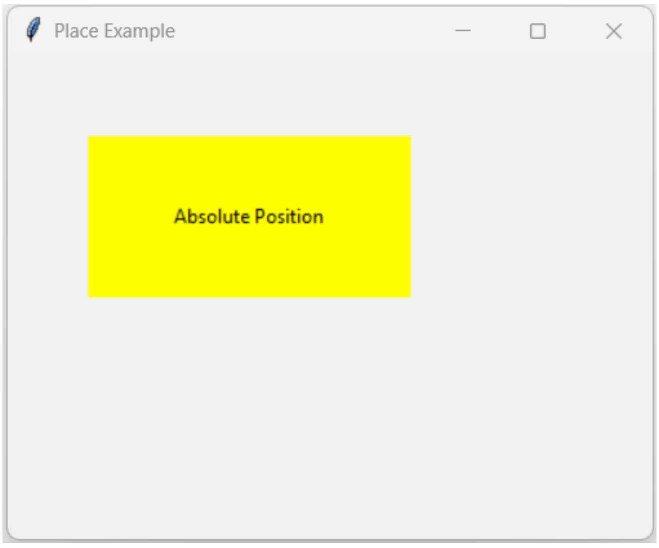
5.4.2 Programs on place() with Options:

The place() method allows you to position widgets at an exact location using coordinates.

- x, y: Specifies the position of the widget using absolute coordinates.
- width, height: Defines the width and height of the widget.

Program:

```
from tkinter import Tk, Label

root = Tk()
root.title("Place Example")
root.geometry("400x300")

label1 = Label(root, text="Absolute Position", bg="yellow", fg="black")
label1.place(x=50, y=50, width=200, height=100)

root.mainloop()
```

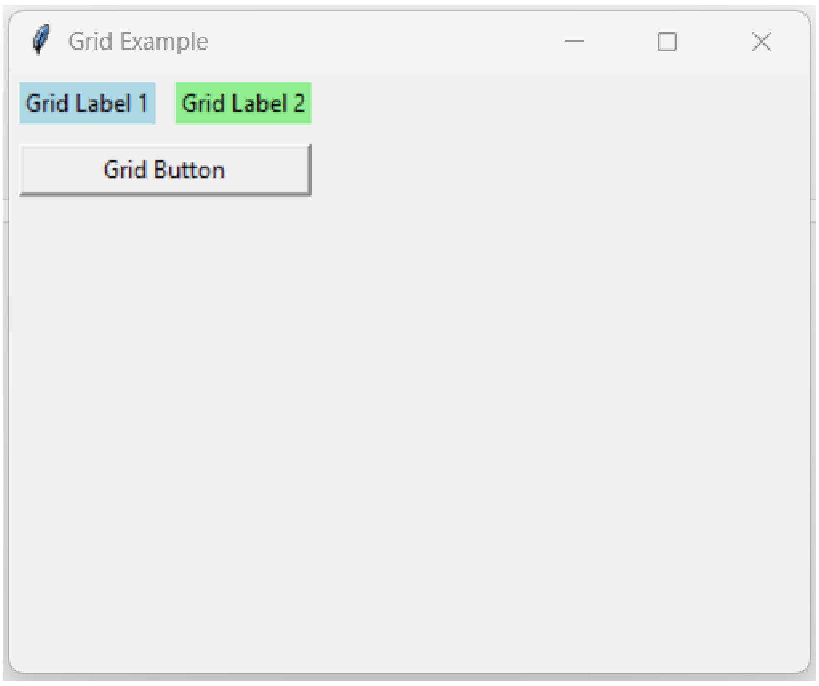


5.4.3 Programs on grid() with Options:

The grid() method organizes widgets in a table-like structure with rows and columns.

- row, column: Specifies the row and column position of the widget.
- rowspan, columnspan: Determines how many rows or columns the widget should span.
- sticky: Aligns the widget within its grid cell (e.g., N, S, E, W).
- padx, pady: Adds padding outside the widget (horizontal and vertical).
- ipadx, ipady: Adds internal padding inside the widget (horizontal and vertical).

```python
from tkinter import Tk, Label, Button

root = Tk()
root.title("Grid Example")
root.geometry("400x300")

label1 = Label(root, text="Grid Label 1", bg="lightblue", fg="black")
label1.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")

label2 = Label(root, text="Grid Label 2", bg="lightgreen", fg="black")
label2.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")

button = Button(root, text="Grid Button")
button.grid(row=1, column=0, columnspan=2, padx=5, pady=5, sticky="nsew")

root.mainloop()
```



5.5 Introduction to Event-Driven Programming:

Event-driven programming is a programming paradigm in which the flow of the program is determined by events—such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs. It is widely used in graphical user interfaces (GUIs) and applications that need to respond to user inputs.

Programs on Button with Command Option and Button States

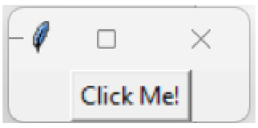
We'll use the Tkinter library, which is the standard Python interface to the Tk GUI toolkit, to illustrate event-driven programming with buttons and their states.

Step-by-Step Guide with Examples:

1. **Setting Up Tkinter**: First, you need to install the Tkinter library if it is not already installed. It usually comes pre-installed with Python.
2. **Creating a Simple Window with a Button**: Let's start by creating a simple window with a button that responds to clicks.

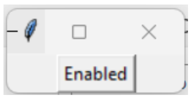
```python
import tkinter as tk

def on_button_click():
    print("Button clicked!")

# Create the main window
root = tk.Tk()
root.title("Event-Driven Programming Example")

# Create a button widget
button = tk.Button(root, text="Click Me!", command=on_button_click)

# Place the button in the window
button.pack()

# Run the application
root.mainloop()
```

```
Button clicked!
```



3. **Adding Command Option to Buttons:** The command option in Tkinter's Button widget allows you to specify a function to be called when the button is clicked. In the example above, on_button_click is called when the button is clicked.
4. **Handling Button States:** Buttons in Tkinter can have different states, such as normal, active, or disabled. You can change the state of a button programmatically.

```python
import tkinter as tk

def toggle_button_state():
    if button['state'] == tk.NORMAL:
        button['state'] = tk.DISABLED
        button.config(text="Disabled")
    else:
        button['state'] = tk.NORMAL
        button.config(text="Enabled")

# Create the main window
root = tk.Tk()
root.title("Button State Example")

# Create a button widget
button = tk.Button(root, text="Enabled", command=toggle_button_state)

# Place the button in the window
button.pack()

# Run the application
root.mainloop()
```

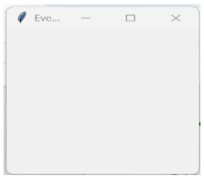


In this example, clicking the button toggles its state between enabled and disabled.

5. **Responding to Various Events**: Tkinter allows you to bind different events to widgets. You can use the bind method to respond to events like key presses, mouse movements, etc.

```python
import tkinter as tk

def on_key_press(event):
    print(f"Key pressed: {event.keysym}")

# Create the main window
root = tk.Tk()
root.title("Event Binding Example")

# Bind key press event to the main window
root.bind('<KeyPress>', on_key_press)

# Run the application
root.mainloop()
```

```
Key pressed: Return
```



5.6 Introduction to Checkbutton and Radiobutton:

1. Checkbutton:

- A Checkbutton is a widget that allows the user to select or deselect a particular option. You can have multiple Checkbuttons, and each can be selected independently.
- Use cases include toggling settings, selecting multiple items from a list, etc.

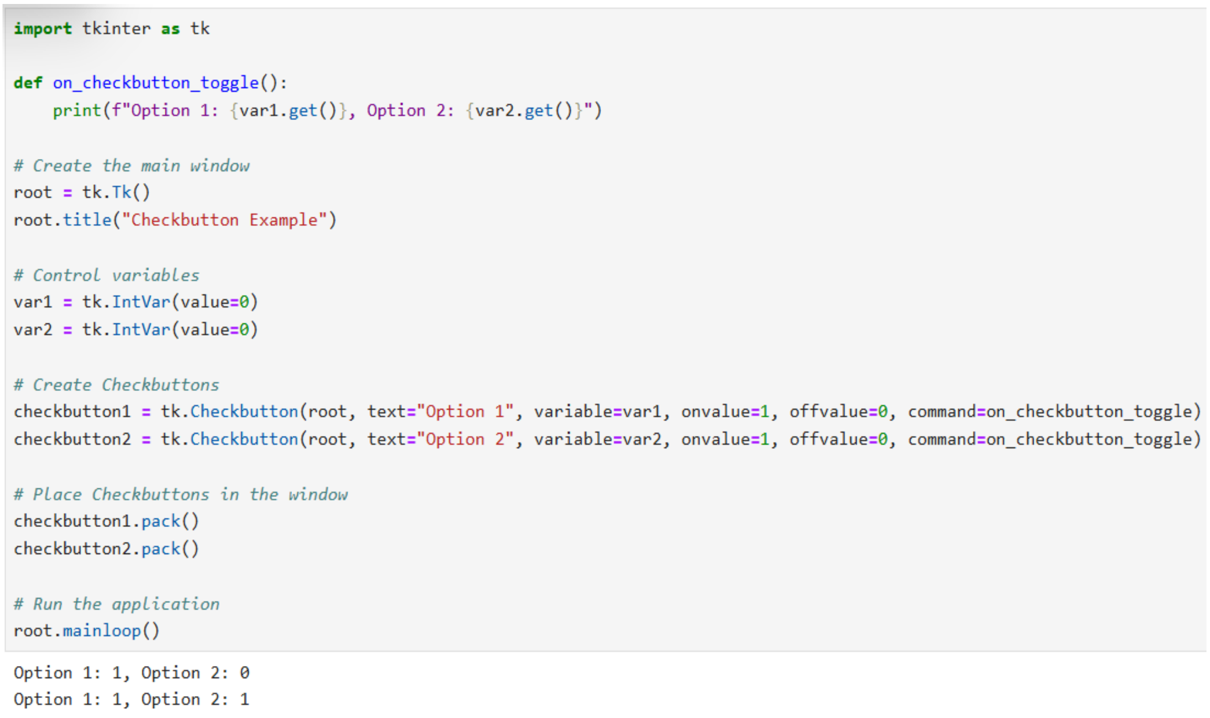
2. Radiobutton:

- A Radiobutton is a widget that allows the user to select one option from a set of mutually exclusive options. Selecting a Radiobutton deselects any other Radiobutton in the same group.
- Use cases include selecting a single option from a list of choices, such as gender selection, quiz options, etc.

5.6.1 Checkbutton Options:

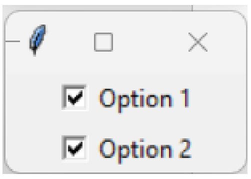
- **text:** The text to be displayed on the Checkbutton.
- **command:** A function to be called when the Checkbutton is toggled.
- **variable:** A control variable (usually tk.IntVar) that holds the state of the Checkbutton (1 if selected, 0 if not).
- **onvalue:** The value to set the variable when the Checkbutton is selected.
- **offvalue:** The value to set the variable when the Checkbutton is deselected.
- **get() method:** Retrieves the current value of the variable.

Program with Checkbuttons demonstrating these options:



```python
import tkinter as tk

def on_checkbutton_toggle():
    print(f"Option 1: {var1.get()}, Option 2: {var2.get()}")

# Create the main window
root = tk.Tk()
root.title("Checkbutton Example")

# Control variables
var1 = tk.IntVar(value=0)
var2 = tk.IntVar(value=0)

# Create Checkbuttons
checkbutton1 = tk.Checkbutton(root, text="Option 1", variable=var1, onvalue=1, offvalue=0, command=on_checkbutton_toggle)
checkbutton2 = tk.Checkbutton(root, text="Option 2", variable=var2, onvalue=1, offvalue=0, command=on_checkbutton_toggle)

# Place Checkbuttons in the window
checkbutton1.pack()
checkbutton2.pack()

# Run the application
root.mainloop()
```

```
Option 1: 1, Option 2: 0
Option 1: 1, Option 2: 1
```



In the above example:

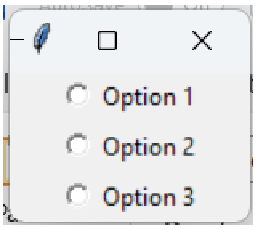
- Two Checkbuttons are created with text, variable, onvalue, and offvalue options.
- The on_checkbutton_toggle function prints the state of both Checkbuttons whenever they are toggled.
- The get() method of the control variables (var1 and var2) is used to retrieve their current values.

5.6.2 Radiobutton Options:

- **text:** The text to be displayed on the Radiobutton.
- **variable:** A control variable (usually tk.IntVar) that holds the state of the selected Radiobutton.
- **value:** The value to set the variable when the Radiobutton is selected.
- **command:** A function to be called when the Radiobutton is selected.
- **get() method:** Retrieves the current value of the variable.

Program with Radiobuttons demonstrating these options:

```python
import tkinter as tk

def on_radiobutton_select():
    print(f"Selected option: {var.get()}")

# Create the main window
root = tk.Tk()
root.title("Radiobutton Example")

# Control variable
var = tk.IntVar(value=0)

# Create Radiobuttons
radiobutton1 = tk.Radiobutton(root, text="Option 1", variable=var, value=1, command=on_radiobutton_select)
radiobutton2 = tk.Radiobutton(root, text="Option 2", variable=var, value=2, command=on_radiobutton_select)
radiobutton3 = tk.Radiobutton(root, text="Option 3", variable=var, value=3, command=on_radiobutton_select)

# Place Radiobuttons in the window
radiobutton1.pack()
radiobutton2.pack()
radiobutton3.pack()

# Run the application
root.mainloop()
```



In the above example:

- Three Radiobuttons are created with text, variable, value, and command options.
- The on_radiobutton_select function prints the currently selected option whenever a Radiobutton is selected.
- The get() method of the control variable (var) is used to retrieve the current value.

5.7 Introduction to Menu and Menubutton widgets:

1. Menu:

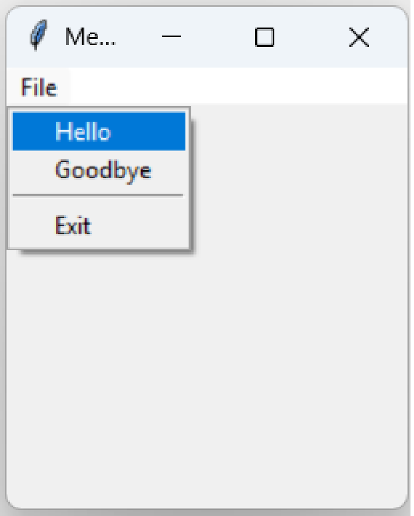
- A Menu in Tkinter is a dropdown list of options or commands that a user can select. It is typically found in a menu bar at the top of a window.
- Menus are used to organize commands and functions in an application, making them easily accessible to the user.

2. Menubutton:

- A Menubutton is a widget that displays a menu when clicked. It is similar to a standard button but is designed to show a dropdown menu.
- Menubuttons can be used to create context menus or standalone menus within a Tkinter application.

5.7.1 Program on Menu Creation:

```python
import tkinter as tk

def hello():
    print("Hello, World!")

def goodbye():
    print("Goodbye, World!")

# Create the main window
root = tk.Tk()
root.title("Menu Example")

# Create a menu bar
menu_bar = tk.Menu(root)

# Create a File menu
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="Hello", command=hello)
file_menu.add_command(label="Goodbye", command=goodbye)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)

# Add the File menu to the menu bar
menu_bar.add_cascade(label="File", menu=file_menu)

# Configure the window to display the menu bar
root.config(menu=menu_bar)

# Run the application
root.mainloop()
```



```
Hello, World!
```

Explanation:

- tk.Menu() is a constructor of the Menu class that initializes a new menu object.
- The methods of this Menu object, such as add_command(), add_cascade(), or add_separator(), are used to add items or submenus to the menu.
- So, while Menu **is a class**, the functions you call to interact with it (like add_command) are **methods** of the Menu class.

Methods:

- add_command(): Adds a menu item that triggers a command.
- add_cascade(): Adds a submenu to the menu.
- add_separator(): Adds a separator line to the menu.

command parameter:

- The command parameter in Tkinter's Button widget is used to specify a function (callback) that should be executed when the button is clicked. This makes it a crucial part of event-driven programming in Tkinter. Instead of binding events manually, the command parameter allows you to directly associate a button with a specific function that performs a desired action.

Summary:

- A menu bar is created using tk.Menu().
- A "File" menu is created and populated with items using add_command(), add_separator(), and add_cascade().
- The hello and goodbye functions are triggered when the corresponding menu items are selected.
- The menu bar is added to the window using root.config(menu=menu_bar).

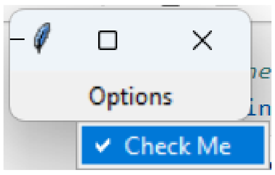
5.7.2 Program to demonstrate on Menubutton with a Checkbutton:

Method:

add_checkbutton(): Adds a checkbutton to the menu.

```
import tkinter as tk

def toggle_option():
    print(f"Option checked: {var.get()}")

# Create the main window
root = tk.Tk()
root.title("Menubutton Example")

# Create a Menubutton
menubutton = tk.Menubutton(root, text="Options")
menubutton.menu = tk.Menu(menubutton, tearoff=0)
menubutton["menu"] = menubutton.menu

# Create a control variable
var = tk.IntVar(value=0)

# Add a checkbutton to the menu
menubutton.menu.add_checkbutton(label="Check Me", variable=var, command=toggle_option)

# Place the Menubutton in the window
menubutton.pack()

# Run the application
root.mainloop()
```

```
Option checked: 1
```



In the above example:

- A Menubutton is created using tk.Menubutton().
- A menu is associated with the Menubutton using menubutton.menu.
- A checkbutton is added to the menu using add_checkbutton(), and it toggles the var variable when clicked.
- The toggle_option function prints the state of the checkbutton whenever it is toggled.

5.8 Introduction to Listbox and Scrollbar Widgets:

1. Listbox:

A Listbox widget displays a list of items from which a user can select one or more items. It is useful for showing lists of data, such as file names, options, etc.

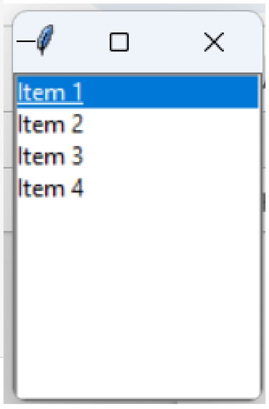
2. Scrollbar:

A Scrollbar widget provides a vertical or horizontal scroll bar for various widgets, such as Listbox, Text, Canvas, etc. It is used to navigate through a list of items or large data sets.

5.8.1 Program on Listbox Widget

```python
import tkinter as tk

def on_select(event):
    selected_item = listbox.get(listbox.curselection())
    print(f"Selected item: {selected_item}")

# Create the main window
root = tk.Tk()
root.title("Listbox Example")

# Create a Listbox
listbox = tk.Listbox(root)

# Insert items into the Listbox
for item in ["Item 1", "Item 2", "Item 3", "Item 4"]:
    listbox.insert(tk.END, item)

# Bind the select event to the Listbox
listbox.bind('<<ListboxSelect>>', on_select)

# Place the Listbox in the window
listbox.pack()

# Run the application
root.mainloop()
```

```
Selected item: Item 1
```



Methods:

- **insert():** Adds items to the Listbox.
- **bind():** Attaches event handlers to Listbox events.
- **get():** Retrieves items from the Listbox.

In the above example:

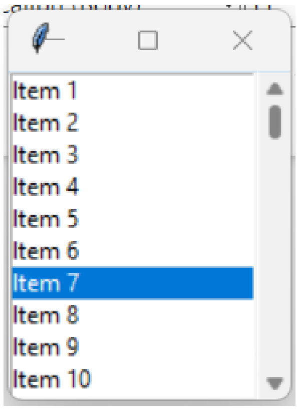
- The insert() method adds items to the Listbox.
- The bind() method attaches the on_select function to the <<ListboxSelect>> event.
- The get() method retrieves the selected item from the Listbox.

5.8.2 Program on Scrollbar Widget:

Options:

- **yscrollcommand:** Links the Scrollbar to the vertical scrolling of another widget, such as a Listbox.
- **command:** Links the Scrollbar to a command that will be called when the Scrollbar is moved.

```python
import tkinter as tk

# Create the main window
root = tk.Tk()
root.title("Scrollbar Example")

# Create a Listbox with a Scrollbar
listbox = tk.Listbox(root)
scrollbar = tk.Scrollbar(root, orient=tk.VERTICAL, command=listbox.yview)
listbox.config(yscrollcommand=scrollbar.set)

# Insert items into the Listbox
for i in range(100):
    listbox.insert(tk.END, f"Item {i+1}")

# Place the Listbox and Scrollbar in the window
listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Run the application
root.mainloop()
```



```
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10
```

In the above example:

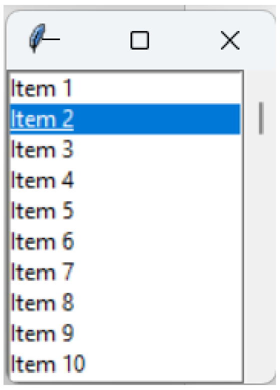
- The command option of the Scrollbar is set to listbox.yview, linking the Scrollbar to the vertical scrolling of the Listbox.
- The yscrollcommand option of the Listbox is set to scrollbar.set, linking the Listbox to the Scrollbar.

5.8.3 Link a Listbox with a Scrollbar Widget:

Combining the Listbox and Scrollbar Widget, we can create a Listbox with a linked Scrollbar.

```python
import tkinter as tk

def on_select(event):
    selected_item = listbox.get(listbox.curselection())
    print(f"Selected item: {selected_item}")

# Create the main window
root = tk.Tk()
root.title("Listbox with Scrollbar Example")

# Create a Listbox with a Scrollbar
listbox = tk.Listbox(root)
scrollbar = tk.Scrollbar(root, orient=tk.VERTICAL, command=listbox.yview)
listbox.config(yscrollcommand=scrollbar.set)

# Insert items into the Listbox
for i in range(100):
    listbox.insert(tk.END, f"Item {i+1}")

# Bind the select event to the Listbox
listbox.bind('<<ListboxSelect>>', on_select)

# Place the Listbox and Scrollbar in the window
listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Run the application
root.mainloop()
```

```
Selected item: Item 2
```



Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

In the above combined example:

- The Listbox and Scrollbar are linked to provide a scrollable list of items.
- The on_select function prints the selected item when an item is selected from the Listbox.

5.9 Introduction to messagebox and Toplevel Widgets:

1. messagebox:

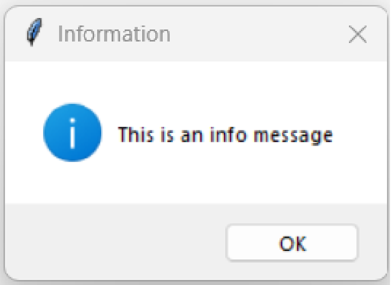
- The messagebox module in Tkinter provides simple pop-up dialogs that display messages to the user. It includes various types of message boxes like information, warning, and error messages.
- These dialogs are commonly used to alert the user or provide information.

2. Toplevel:

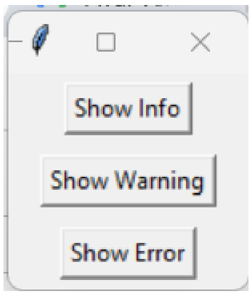
- The Toplevel widget in Tkinter creates a new top-level window. It is used to create additional windows apart from the main application window.
- It can be used to create dialog boxes, popup windows, and other secondary windows.

5.9.1 Program on messagebox Widget:

```python
import tkinter as tk
from tkinter import messagebox

def show_info():
    messagebox.showinfo("Information", "This is an info message")

def show_warning():
    messagebox.showwarning("Warning", "This is a warning message")

def show_error():
    messagebox.showerror("Error", "This is an error message")

# Create the main window
root = tk.Tk()
root.title("Messagebox Example")

# Create buttons to trigger message boxes
info_button = tk.Button(root, text="Show Info", command=show_info)
warning_button = tk.Button(root, text="Show Warning", command=show_warning)
error_button = tk.Button(root, text="Show Error", command=show_error)

# Place the buttons in the window
info_button.pack(pady=5)
warning_button.pack(pady=5)
error_button.pack(pady=5)

# Run the application
root.mainloop()
```


Information

This is an info message



Methods:

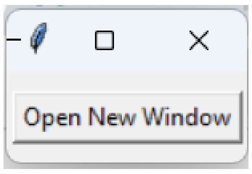
- showinfo(): Displays an informational message box.
- showwarning(): Displays a warning message box.
- showerror(): Displays an error message box.

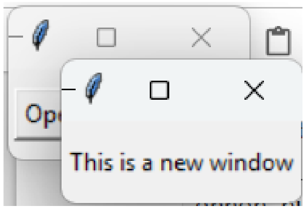
In the above example:

- show_info(), show_warning(), and show_error() functions display informational, warning, and error message boxes respectively.
- Buttons are created to trigger these functions, and they display corresponding message boxes when clicked.

5.9.2 Program on Toplevel:

```python
import tkinter as tk

def open_new_window():
    new_window = tk.Toplevel(root)
    new_window.title("New Window")
    label = tk.Label(new_window, text="This is a new window")
    label.pack(pady=10)

# Create the main window
root = tk.Tk()
root.title("Toplevel Example")

# Create a button to open a new window
open_button = tk.Button(root, text="Open New Window", command=open_new_window)
open_button.pack(pady=10)

# Run the application
root.mainloop()
```





In the above example:

- The open_new_window() function creates a new top-level window using the Toplevel widget.
- A label is added to the new window to display some text.
- A button in the main window triggers the open_new_window() function, opening the new window.

5.10 filedialog widget:

The filedialog widget in Tkinter is a module provides pre-built widgets that allow users to interact with the file system for actions like:

- Selecting files to open.
- Specifying file paths to save files.
- Navigating directories.

Key Features of the filedialog Module:

- **File Selection:** Helps users browse and select files for opening or saving.
- **Cross-Platform:** Offers dialogs that match the appearance and behavior of the host operating system.
- **Customizable Options:** Allows developers to configure filters for file types, default directory locations, and titles.

Filedialog widget Functions:

1. **askopenfile():** Opens a dialog to select a file and returns a file object (opened in read mode by default).
2. **askopenfiles():** Opens a dialog to allows the selection of multiple files and returns a list of file objects.
3. **askopenfilename():** Opens a dialog for selecting a file to open. Returns the selected file's path.
4. **askopenfilenames():** Opens a dialog to allows the selection of multiple files and returns a tuple of file paths.
5. **askdirectory():** Opens a dialog for selecting a directory. Returns the selected directory's path.
6. **asksaveasfile():** Opens a "Save As" dialog, and allows the user to save data into a new file and returns a file object.
7. **asksaveasfilename():** Opens a dialog for selecting a file to save. Returns the selected file's path.

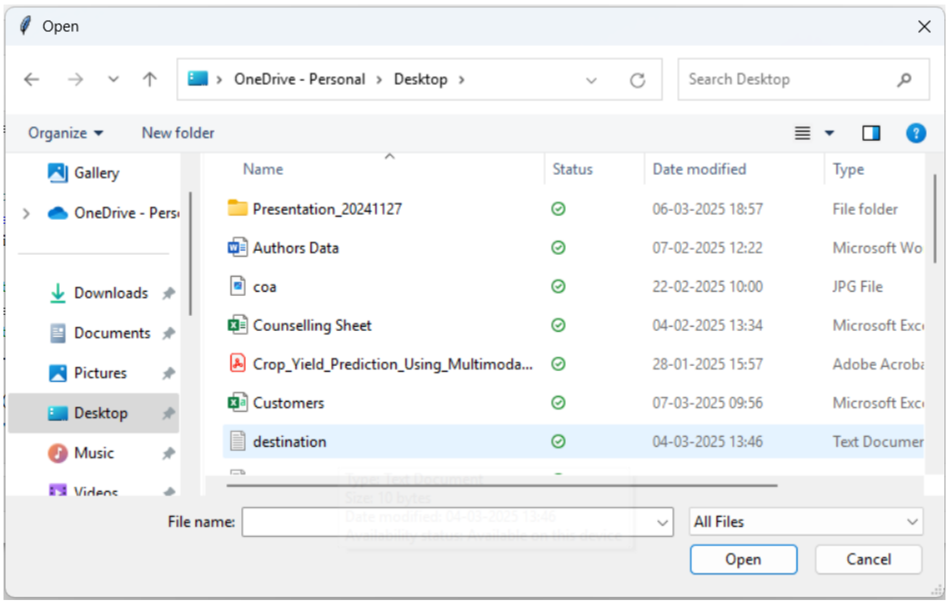
5.10.1 filedialog methods:

1. askopenfile()

Allows users to select a file and returns a file object (opened in read mode by default).

```
import tkinter as tk
from tkinter import filedialog

# Function to select a file and display its content
def open_file():
    file = filedialog.askopenfile(mode="r", filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
    if file:
        print(f"File Name: {file.name}")
        content = file.read()
        print(f"Content: {content}")
        file.close()

root = tk.Tk()
root.withdraw()  # Hide the root window
open_file()
```

```
File Name: C:/Users/K LEELA PRASAD/OneDrive/Desktop/destination.txt
Content: Tomorrow
```



Name	Status	Date modified	Type
Presentation_20241127	✓	06-03-2025 18:57	File folder
Authors Data	✓	07-02-2025 12:22	Microsoft Wo
coa	✓	22-02-2025 10:00	JPG File
Counselling Sheet	✓	04-02-2025 13:34	Microsoft Exc
Crop_Yield_Prediction_Using_Multimoda...	✓	28-01-2025 15:57	Adobe Acrob
Customers	✓	07-03-2025 09:56	Microsoft Exc
destination	✓	04-03-2025 13:46	Text Documer

Explanation to the above program:

Import Statements:

1. import tkinter as tk This imports the tkinter library, which is Python's standard GUI (Graphical User Interface) toolkit. The alias tk is used to shorten the reference to the module.
2. from tkinter import filedialog This imports the filedialog module from tkinter. filedialog provides functions for creating file dialog boxes, such as opening or saving files.

Function Definition:

3. Function to select a file and display its content A comment explaining what the upcoming function does.
4. def open_file(): This defines a function named open_file. This function allows the user to select a file and displays its content.
5. file = filedialog.askopenfile(mode="r", filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
    - filedialog.askopenfile: Opens a dialog box for the user to select a file.
    - mode="r": Opens the selected file in read mode ("r").
    - filetypes = [("Text Files", "*.txt"), ("All Files", "*.*")]: Filters the types of files shown in the dialog box.
        - "Text Files", "*.txt": Only files with the .txt extension are displayed.
        - "All Files", "*.*": Displays all types of files. The function returns a file object, which is assigned to the variable file. If the user cancels the operation, file will be None.
6. if file: This checks if the user selected a file (i.e., file is not None).
7. print(f"File Name: {file.name}") Prints the name of the selected file using file.name.
8. content = file.read() Reads the entire content of the file and assigns it to the variable content.
9. print(f"Content: {content}") Prints the content of the file to the console.
10. file.close() Closes the file to free up system resources.

Create and Configure the Tkinter Window:

11. root = tk.Tk() Creates the main Tkinter window and assigns it to the variable root.
12. root.withdraw() Hides the main Tkinter window so that it doesn't appear on the screen. This is done because the program only needs to display the file dialog box, not the main window.

Function Call:

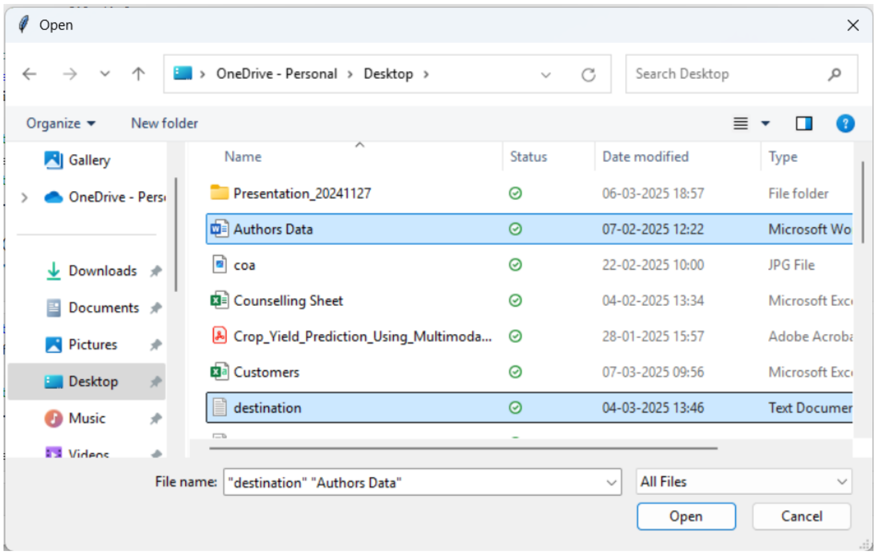
13. open_file() Calls the open_file function to execute the file selection and content display process.

2. askopenfiles()

Allows the selection of multiple files and returns a list of file objects.

```
def open_multiple_files():
    files = filedialog.askopenfiles(filetypes=[("Python Files", "*.py"), ("All Files", "*.*")])
    for file in files:
        print(f"File Name: {file.name}")
        file.close()

open_multiple_files()
```

```
File Name: C:/Users/K LEELA PRASAD/OneDrive/Desktop/Authors Data.docx
File Name: C:/Users/K LEELA PRASAD/OneDrive/Desktop/destination.txt
```



Name	Status	Date modified	Type
Presentation_20241127	✓	06-03-2025 18:57	File folder
Authors Data	✓	07-02-2025 12:22	Microsoft Wo
coa	✓	22-02-2025 10:00	JPG File
Counselling Sheet	✓	04-02-2025 13:34	Microsoft Exc
Crop_Yield_Prediction_Using_Multimoda...	✓	28-01-2025 15:57	Adobe Acroba
Customers	✓	07-03-2025 09:56	Microsoft Exc
destination	✓	04-03-2025 13:46	Text Documer

File name: "destination" "Authors Data"   All Files

Explanation:

Function Definition:

1. def open_multiple_files(): This defines a function named open_multiple_files. The purpose of this function is to allow the user to select multiple files and print their names.

File Dialog and File Selection:

2. files = filedialog.askopenfiles(filetypes=[("Python Files", "*.py"), ("All Files", "*.*")])
   - filedialog.askopenfiles: Opens a file dialog box that allows the user to select multiple files. It returns a list of file objects corresponding to the selected files.
   - filetypes=[("Python Files", "*.py"), ("All Files", "*.*")]: Filters the types of files shown in the dialog box.
     - "Python Files", "*.py": Only files with the .py extension (Python files) are shown.
     - "All Files", "*.*": Displays all files irrespective of their extensions.

Iterating Through Selected Files:

3. for file in files: This creates a loop to iterate over each file object in the files list.

4. print(f"File Name: {file.name}")
    - Prints the name of each selected file using the file.name attribute.
    - f"File Name: {file.name}": A formatted string (f-string) that dynamically inserts the file name into the output.
5. file.close() Closes each file after its name has been printed. This is important for freeing up system resources associated with the file object.

Calling the Function:

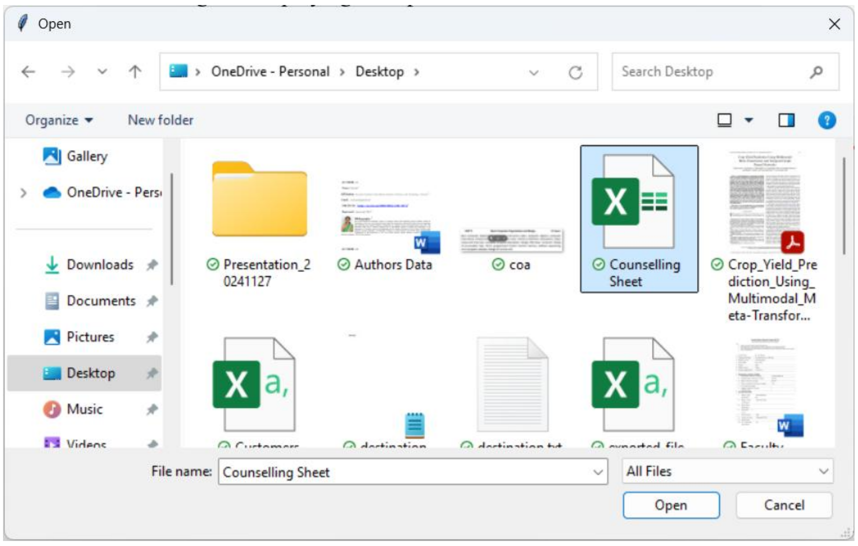
6. open_multiple_files() This calls the open_multiple_files function to execute the process of selecting and displaying multiple files.

3. askopenfilename()

Returns the file path (string) of a selected file.

```python
def get_file_path():
    file_path = filedialog.askopenfilename(filetypes=[("Image Files", "*.png;*.jpg;*.jpeg"), ("All Files", "*.*")])
    print(f"Selected File Path: {file_path}")

get_file_path()
```

```
Selected File Path: C:/Users/K LEELA PRASAD/OneDrive/Desktop/Counselling Sheet.xlsx
```



Explanation:

Function Definition:

1. def get_file_path(): This defines a function named get_file_path. Its purpose is to open a dialog box for file selection and display the file path of the selected file.

File Dialog and File Path Selection:

2. file_path = filedialog.askopenfilename(filetypes=[("Image Files", "*.png;*.jpg;*.jpeg"), ("All Files", "*.*")])
    - filedialog.askopenfilename: Opens a file selection dialog box that allows the user to choose a single file.

- filetypes=[("Image Files", "*.png;*.jpg;*.jpeg"), ("All Files", "*.*")]: Filters the files displayed in the dialog box:
    - "Image Files", "*.png;*.jpg;*.jpeg": Shows only image files with .png, .jpg, or .jpeg extensions.
    - "All Files", "*.*": Displays all types of files, regardless of their extensions.
- The file path of the selected file is returned as a string and stored in the variable file_path. If the user cancels the operation, file_path will be an empty string.

Print the Selected File Path:

3. print(f"Selected File Path: {file_path}")
    - Prints the file path of the selected file using an f-string (formatted string).
    - If no file is selected, the printed message will show an empty string as the file path.

Calling the Function:

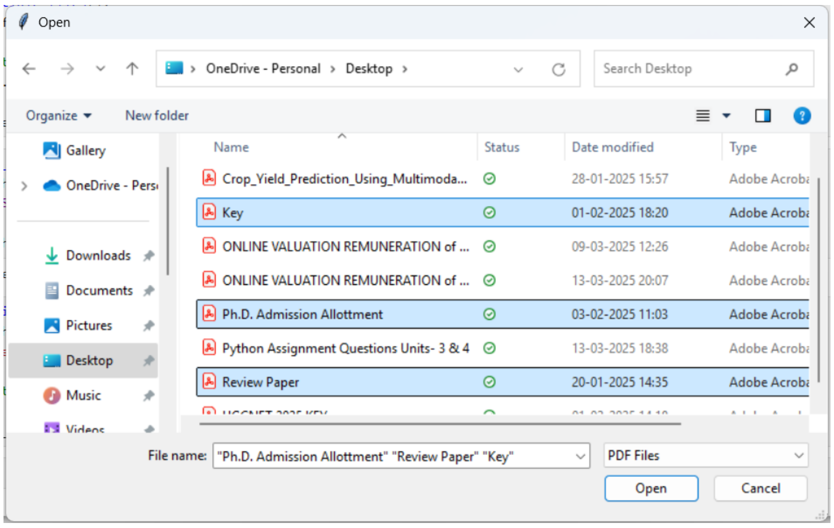
4. get_file_path() Calls the get_file_path function to execute the process of opening the file dialog and displaying the selected file's path.

4. askopenfilenames()

Allows the selection of multiple files and returns a tuple of file paths.

```python
def get_multiple_file_paths():
    file_paths = filedialog.askopenfilenames(filetypes=[("PDF Files", "*.pdf"), ("All Files", "*.*")])
    print("Selected Files:")
    for path in file_paths:
        print(path)

get_multiple_file_paths()
```

```
Selected Files:
C:/Users/K LEELA PRASAD/OneDrive/Desktop/Key.pdf
C:/Users/K LEELA PRASAD/OneDrive/Desktop/Ph.D. Admission Allottment.pdf
C:/Users/K LEELA PRASAD/OneDrive/Desktop/Review Paper.pdf
```



Name	Status	Date modified	Type
Crop_Yield_Prediction_Using_Multimoda...	✓	28-01-2025 15:57	Adobe Acroba
Key	✓	01-02-2025 18:20	Adobe Acroba
ONLINE VALUATION REMUNERATION of ...	✓	09-03-2025 12:26	Adobe Acroba
ONLINE VALUATION REMUNERATION of ...	✓	13-03-2025 20:07	Adobe Acroba
Ph.D. Admission Allottment	✓	03-02-2025 11:03	Adobe Acroba
Python Assignment Questions Units- 3 & 4	✓	13-03-2025 18:38	Adobe Acroba
Review Paper	✓	20-01-2025 14:35	Adobe Acroba

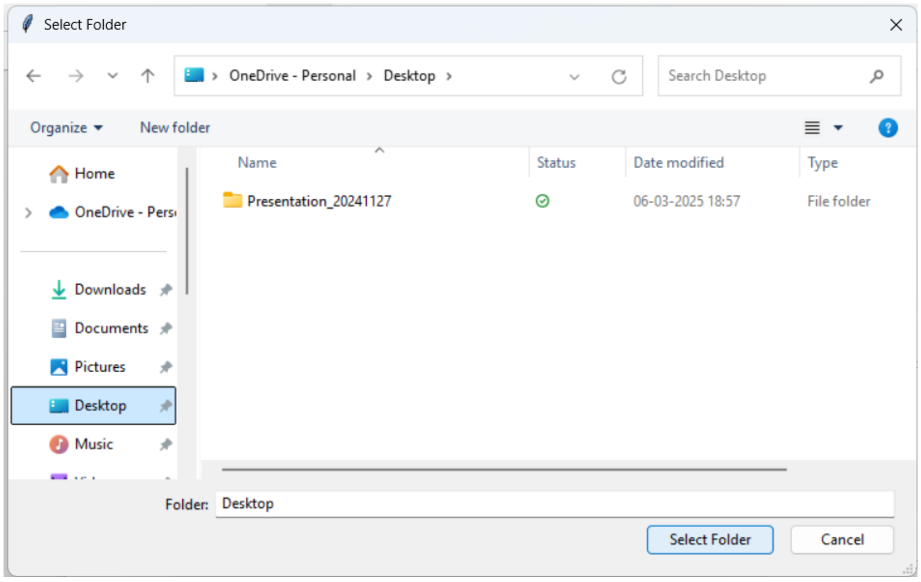
File name: "Ph.D. Admission Allottment" "Review Paper" "Key" PDF Files

5. askdirectory()

This function opens a directory selection dialog and returns the selected directory's path.

```
import tkinter as tk
from tkinter import filedialog

def select_directory():
    directory_path = filedialog.askdirectory()
    print(f"Selected Directory: {directory_path}")

select_directory()
```

```
Selected Directory: C:/Users/K LEELA PRASAD/OneDrive/Desktop
```



Name | Status | Date modified | Type
Presentation_20241127 | | 06-03-2025 18:57 | File folder

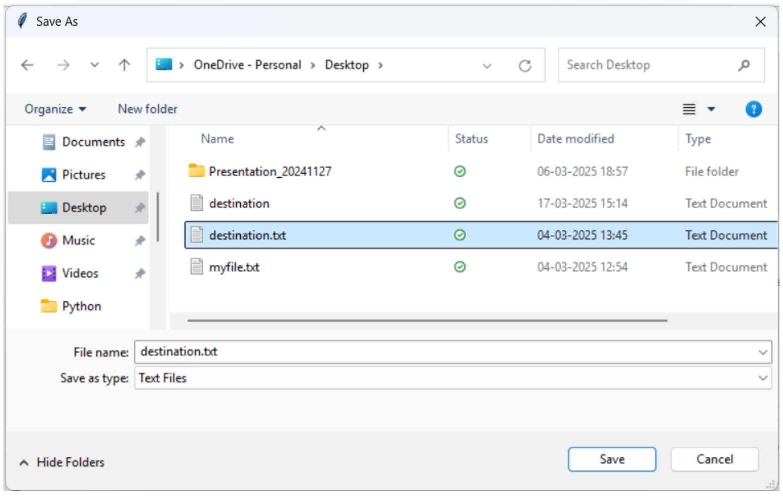
Folder: Desktop

6. **asksaveasfile()**

This function opens a "Save As" dialog, and allows the user to save data into a new file and returns a file object.

```
def save_file():
    file = filedialog.asksaveasfile(mode="w", defaultextension=".txt", filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
    if file:
        file.write("Hello, this is a saved file!")
        print(f"File Saved: {file.name}")
        file.close()

save_file()
```

```
File Saved: C:/Users/K LEELA PRASAD/OneDrive/Desktop/destination.txt
```



Name	Status	Date modified	Type
Presentation_20241127	✓	06-03-2025 18:57	File folder
destination	✓	17-03-2025 15:14	Text Document
destination.txt	✓	04-03-2025 13:45	Text Document
myfile.txt	✓	04-03-2025 12:54	Text Document

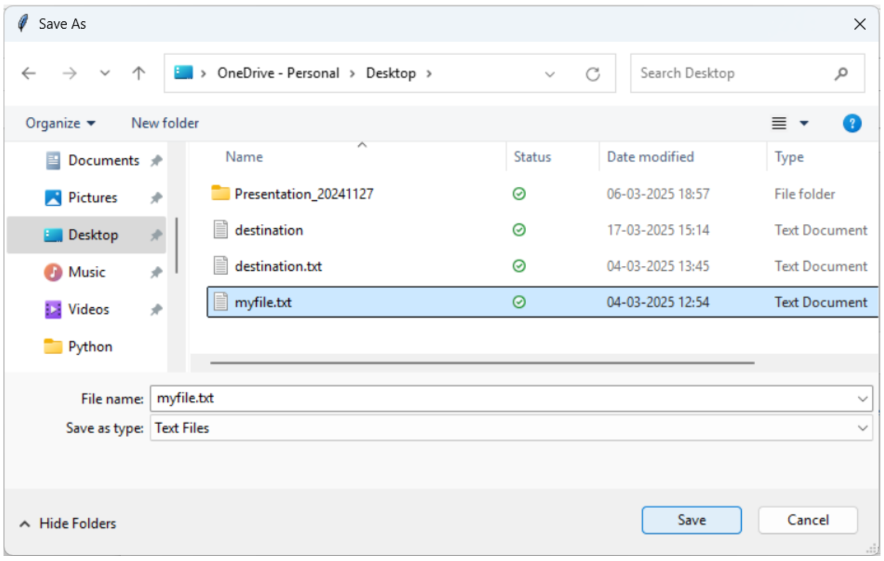
File name: destination.txt
Save as type: Text Files

7. **asksaveasfilename()**

This function is similar to asksaveasfile(), but it only returns the file path as a string, without opening or creating the file.

```
def save_file_as():
    file_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")])
    print(f"File to Save: {file_path}")

save_file_as()
```

```
File to Save: C:/Users/K LEELA PRASAD/OneDrive/Desktop/myfile.txt.txt
```



Name	Status	Date modified	Type
Presentation_20241127	✓	06-03-2025 18:57	File folder
destination	✓	17-03-2025 15:14	Text Document
destination.txt	✓	04-03-2025 13:45	Text Document
myfile.txt	✓	04-03-2025 12:54	Text Document

File name: myfile.txt
Save as type: Text Files

Appendix

What is Event-Driven Programming?

Event-driven programming is a programming paradigm where the flow of the program is determined by events, such as user actions (clicks, keypresses, etc.), messages, or changes in the system's state. In this approach, the program remains idle, waiting for an event to occur. When an event happens, the corresponding code (often referred to as a "callback" or "event handler") is executed to respond to it.

This paradigm is particularly useful for creating interactive applications, where user interactions drive the application's behavior.

Significance of Event-Driven Programming in GUI-Based Applications

In graphical user interface (GUI) applications, event-driven programming is crucial for creating a responsive and interactive user experience. Here’s why:

- **User Interaction:** GUI applications rely on user input such as button clicks, mouse movements, and text input. Event-driven programming enables the application to respond dynamically to these actions.
- **Asynchronous Execution:** It allows the application to handle multiple events concurrently, ensuring that the GUI remains responsive even when performing background tasks.
- **Scalability:** Complex applications with multiple interactive elements can be managed efficiently by associating specific event handlers with corresponding events.
- **Decoupling:** Logic and interface components are separated, as event handlers focus on specific tasks. This improves code organization and maintainability.
    - **For example**, in a word processing application, pressing a key triggers an event that inserts the character into the document. Similarly, clicking the "Save" button triggers an event to save the file.

How Tkinter Uses Event-Driven Principles

Tkinter, Python's standard GUI library, is built around the event-driven programming model. The Tkinter mainloop continuously monitors for events and dispatches them to the appropriate event handlers.

1. Main Event Loop:

- The mainloop() function initiates the event-driven mechanism. It keeps the application running, waiting for events (e.g., clicks, keypresses, window resizing) and dispatching them when they occur.

2. Callback Functions:

- Widgets like Buttons, Entries, or Menus in Tkinter use callback functions to handle specific events. For instance, a Button widget can have a command parameter that specifies a function to be called when the button is clicked.

3. Event Binding:

- Tkinter allows fine-grained control over events through the .bind() method. This lets developers explicitly associate specific events (e.g., <Button-1> for left mouse clicks or <KeyPress> for keypresses) with corresponding handlers.

Assignment Questions

Tkinter GUI and Event-Driven Programming

1. Explain the differences between terminal-based programs and GUI-based programs in terms of user interaction and application design. How does Tkinter facilitate GUI development in Python?
2. What is event-driven programming? Discuss its significance in GUI-based applications and explain how Tkinter uses event-driven principles to manage user interactions.

Tkinter Widgets: Label, Entry, Button

3. Discuss the purposes of Label, Entry, and Button widgets in Tkinter. Provide examples to illustrate how each of these widgets is created and used in a Tkinter application.
4. Explain the use of the command parameter in Tkinter Button widgets. How can it be used to trigger functions in response to user actions? Provide a practical example.

Tkinter Geometry Methods: pack(), grid(), place()

5. Compare and contrast the pack(), grid(), and place() geometry management methods in Tkinter. When should each method be used, and what are their advantages and limitations?
6. Design a Tkinter application with a layout that demonstrates the combined use of pack(), grid(), and place(). Explain the rationale for your chosen layout.

Command Buttons and Event Handling

7. What are command buttons in Tkinter, and how do they interact with events? Discuss the use of callback functions to handle button clicks in a Tkinter application.
8. Explain the process of binding events to widgets in Tkinter. How does the .bind() method differ from the command parameter? Provide an example to illustrate your explanation.

CheckButton and Radiobutton Widgets

9. Explain the functionality of CheckButton and Radiobutton widgets in Tkinter. How are they different, and what are their typical use cases?
10. Discuss how you can group Radiobuttons in Tkinter to ensure that only one option is selected at a time. Illustrate with an example.

Menu and MenuButton Widgets

11. Describe how menus and menubuttons are created in a Tkinter application. What are their purposes, and how can they enhance user experience? Provide a use case.
12. Explain the concept of cascading menus in Tkinter. How can you implement a multi-level menu using the Menu widget?

Listbox and Scrollbar Widgets

13. How are Listbox and Scrollbar widgets used together in Tkinter? Write a Python program that demonstrates a scrollable list of items using these widgets.

14. Discuss how the yscrollcommand and xscrollcommand parameters work in conjunction with Scrollbar widgets to provide scrolling functionality in Tkinter applications.

MessageBox and Toplevel Widgets

15. What is the role of the MessageBox widget in Tkinter applications? How can it be used to display alerts, confirmations, or information to users? Provide an example of its usage.

16. Explain the purpose of the Toplevel widget in Tkinter. Discuss how it differs from the main application window and how it can be utilized to create dialogs or additional windows.

File Dialog Widget

17. What is the purpose of the File Dialog widget in Tkinter? Illustrate how to use askopenfilename and asksaveasfilename methods to allow users to select files in a GUI application.

18. Create a Tkinter application that utilizes a File Dialog widget to load and display the contents of a text file. Explain the steps involved in implementing this functionality.