

Contents

1. Python Tkinter:	2
Why do we need Tkinter?	2
2. Create First Tkinter GUI Application	2
1. Tk()	2
2. mainloop()	3
3. What are Widgets in Tkinter?	4
3.1. Tkinter Widget Basics	5
3.2. Tkinter Widget Intermidate	6
3.3. Tkinter Widget Advance	6
Example: guiApp3.py	6
4. Common Tkinter Widget Properties.....	7
Example: widgetWithProperties.py	8
5. Geometry Management	9
5.1. Types of Geometry Managers	9
5.1.1. pack() — The Packing Manager	10
Common options:	10
5.1.2. grid() — The Grid Manager	10
Common options:	10
5.1.3. place() — The Placement Manager	11
Common options:	11
5.2. Comparison Summary.....	11
5.3. Nested Layouts (Combination Technique)	12
5.4. Geometry Management Hierarchy.....	13
In Summary	13
6. Example:	13
Example 1: Login Form without event-handling. (imp)	13
Output:	14
Example 2: Login Form with Event Handling	15
Example 3: Student Registration Form without event-handling.	16
Output:	17
Example 4: Student Registration Form with event-handling.	18
Output:	20

1. Python Tkinter:

- **Tkinter** is Python's built-in library for creating graphical user interfaces (GUIs). It acts as a lightweight wrapper around Tk GUI toolkit. It **supports layout management, event handling** and **customization**, making it ideal for rapid GUI development in Python.

Why do we need Tkinter?

1. Provides a **built-in** and **easy-to-use** way to **create GUI** (Graphical User Interface) applications in Python.
2. **Offers various widgets** like buttons, labels, text boxes and menus to build interactive apps.
3. **Eliminates the need for external** GUI frameworks tkinter comes bundled with Python.
4. Useful for building **desktop applications** like calculators, form apps and dashboards.
5. Supports **event-driven programming**, making it suitable for responsive user interfaces.

2. Create First Tkinter GUI Application

To create a Tkinter Python app, follow these basic steps:

1. **Import the tkinter module:** Import the tkinter module, which is necessary for creating the GUI components.
2. **Create the main window (container):** Initialize the main application window using the `Tk()` class.
3. **Set Window Properties:** We can set properties like the title and size of the window.
4. **Add widgets to the main window:** We can add any number of widgets like **buttons, labels, entry fields**, etc., to the main window to design the interface.
5. **Pack Widgets:** Use geometry managers like `pack()`, `grid()` or `place()` to arrange the widgets within the window.
6. **Apply event triggers to the widgets:** We can attach event triggers to the widgets to define how they respond to user interactions.

There are **two main methods** used which user needs to remember while creating the Python application with GUI. These methods are:

1. Tk()

- To create a main window in Tkinter, we use the `Tk()` class.

Syntax:

```
root = Tk(screenName=None, baseName=None, className='Tk', useTk=1)
```

Parameter:

- **screenName (optional)**: Specifies the **display** (mainly used on Unix systems with multiple screens).
- **baseName (optional)**: Sets the base name for the application (default is the script name).
- **className (optional)**: Defines the name of the window class (used for styling and window manager settings).
- **useTk (optional)**: A boolean that tells whether to initialize the Tk subsystem (usually left as default 1).

2. mainloop()

- The `mainloop()` method is **used to run application** once it is ready.
- It is an infinite loop that keeps application running, waits for events to occur (such as button clicks) and processes these events as long as window is not closed.

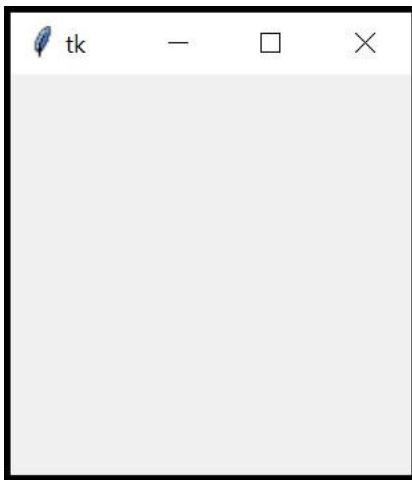
Example: This code **initializes a basic GUI** window using Tkinter. The `tkinter.Tk()` **function** creates main application window, `mainloop()` **method** starts event loop, which keeps the window running and responsive to user interactions.

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

#widgets are added here

m.mainloop()
```

Output:



3. What are Widgets in Tkinter?

- In Tkinter, a widget is essentially a **graphical component** that the user can interact with. They can range from simple elements like **buttons and labels** to **more complex ones** like text entry fields, **listboxes**, and **canvases**.
- Each widget **serves a specific purpose** and can be customized to fit the design and functionality requirements of your application.
- Each widget in Tkinter is **an instance of a specific class** defined in the Tkinter Module.
- These classes provide **methods and attributes** that allow us to **configure the widget's** appearance, behavior, and functionality.
- Widgets are typically **added to the application's window or frames** using **layout managers** like **pack()**, **grid()**, or **place()**, which determine their position and size within the interface.

Example:

```
from tkinter import *

# create root window
root = Tk()

# frame inside root window
frame = Frame(root)

# geometry method
frame.pack()

# button inside frame which is
# inside root
button = Button(frame, text='Submit')
button.pack()

# Tkinter event loop
root.mainloop()
```

3.1. Tkinter Widget Basics

Tkinter supports the below mentioned core widgets -

Widgets	Description / Purpose	Example Code Snippet
<u>Label</u>	Display static text or images. Displays text or image that users can't edit .	<code>tk.Label(root, text="Hello, Tkinter!")</code>
<u>Button</u>	It is used to create/add clickable buttons to our application	<code>tk.Button(root, text="ClickMe", command=my_func)</code>
<u>Entry</u>	It is used to input single line text entry from user	<code>tk.Entry(root, width=25)</code>
Text	Multi-line text input area (for long text).	<code>tk.Text(root, height=5, width=30)</code>
<u>Frame</u>	It is used as container to hold and organize the widgets within a container. A container widget to group other widgets.	<code>tk.Frame(root, bg="lightgray", bd=2, relief="sunken")</code>
<u>RadioButton</u>	It is used to implement one-of-many selection as it allows only one option to be selected (Used to create radio button)	<code>tk.Radiobutton(root, text="Option 1", value=1, variable=var)</code>
<u>Checkbutton</u>	Create checkboxes for boolean options.	<code>tk.Checkbutton(root, text="I agree")</code>
<u>ListBox</u>	Display a list of items for selection.	<code>tk.Listbox(root, height=4)</code>
<u>Scrollbar</u>	Add scrollbars to widgets like Listbox. Adds scrolling capability to widgets (like Text, Canvas, Listbox).	<code>tk.Scrollbar(root, orient="vertical")</code>
<u>Menu</u>	It is used to create all kinds of menu used by an application Creates menu bars and dropdown menus.	<code>menu = tk.Menu(root)</code>
Menubutton	Button that displays a menu when clicked.	<code>tk.Menubutton(root, text="Options")</code>
<u>Canvas</u>	Draw shapes, lines, text, and images. Used for drawing shapes, images, or custom graphics .	Used for drawing shapes, images, or custom graphics .

3.2. Tkinter Widget Intermidate

Widgets	Description	Example Code Snippet
<u>Text</u>	Create a multiline text input with advanced editing capabilities.	<code>tk.Text(root, height=5, width=30)</code>
<u>Combobox</u>	Provide a dropdown list with editable text entry.	<code>ttk.Combobox(root, values=["Red", "Green", "Blue"])</code>
<u>Scale</u>	Create a scale widget for selecting values within a range. Creates a slider to choose numeric values.	<code>tk.Scale(root, from_=0, to=100, orient="horizontal")</code>
<u>Toplevel</u>	Create additional windows/dialogs.	<code>tk.Toplevel(root)</code>
<u>Message</u>	Display simple messages or notifications. Displays multi-line text like Label but wraps automatically.	<code>tk.Message(root, text="Welcome to Tkinter!")</code>
<code>ttk.Progressbar</code>	Displays progress of a task visually.	<code>ttk.Progressbar(root, length=200, mode="determinate")</code>
<u>Spinbox</u>	Provide a numerical input with up/down arrows. Numeric input field with increment/decrement arrows	<code>tk.Spinbox(root, from_=0, to=10)</code>

3.3. Tkinter Widget Advance

Widgets	Description	Example Code Snippet
<u>MessageBox</u> MessageBox (<i>from <code>tkinter.messagebox</code></i>)	Display dialog boxes for messages, warnings, etc. Displays dialog boxes (info, warning, error, etc.).	<code>messagebox.showinfo("Info", "Hello!")</code>

Example: guiApp3.py

```
import tkinter as tk
from tkinter import ttk, messagebox
```

```

def greet():
    name = entry.get()
    messagebox.showinfo("Greeting", f"Hello, {name}!")

root = tk.Tk()
root.title("Basic Tkinter Widgets")

tk.Label(root, text="Enter your name:").pack(pady=5)
entry = tk.Entry(root, width=30)
entry.pack(pady=5)

tk.Button(root, text="Greet", command=greet).pack(pady=5)

tk.Checkbutton(root, text="Subscribe to newsletter").pack()

tk.Radiobutton(root, text="Male", value=1).pack()
tk.Radiobutton(root, text="Female", value=2).pack()

ttk.Combobox(root, values=["Nepal", "India", "China"],
width=27).pack(pady=5)

root.mainloop()

```

4. Common Tkinter Widget Properties

Property	Description	Example
bg or background	Background color of the widget.	bg="lightblue"
fg or foreground	Text color (for widgets that display text).	fg="black"
font	Font type, size, and style.	font=("Arial", 12, "bold")
text	The text displayed on the widget (like Label or Button).	text="Click Me"
width	Width of the widget (in characters or pixels depending on the widget).	width=20
height	Height of the widget (in lines	height=2

	or pixels).	
padx	Extra horizontal padding inside the widget.	padx=10
pady	Extra vertical padding inside the widget.	pady=5
relief	Border style — flat, raised, sunken, groove, ridge.	relief="raised"
borderwidth or bd	Thickness of the border.	bd=2
cursor	Mouse pointer shape when hovering.	cursor="hand2"
state	Widget state: normal, active, or disabled.	state="disabled"
anchor	Position of the text/image inside widget (n, s, e, w, center).	anchor="center"
justify	Text alignment for multiple lines (left, center, right).	justify="left"
image	Display an image instead of (or with) text.	image=photo_obj
compound	Display both text and image together (left, right, top, bottom, center).	compound="left"
takefocus	Determines if widget can take keyboard focus.	takefocus=True
highlightcolor	Color of the focus highlight when widget is active.	highlightcolor="blue"
highlightthickness	Thickness of focus highlight border.	highlightthickness=2

Example: widgetWithProperties.py

```
import tkinter as tk

root = tk.Tk()
root.title("Common Tkinter Properties Example")

btn = tk.Button(
    root,
    text="Click Me!",
    bg="lightblue",
```

```

fg="darkblue",
font=("Arial", 14, "bold"),
relief="raised",
bd=4,
padx=10,
pady=5,
cursor="hand2"
)
btn.pack(pady=10)

root.mainloop()

```

5. Geometry Management

- In **Tkinter**, *geometry management* refers to the **process of controlling how widgets are arranged (positioned and sized)** inside their parent container (like a window or frame).
- Every widget in Tkinter — such as `Label`, `Button`, `Entry`, etc. must be **placed on the screen**.
Tkinter uses **geometry managers** to handle that placement automatically.

Without geometry management:

- Widgets would overlap or be misplaced.
- GUI design would break when the window is resized.
- Developers would need to manually calculate every widget's position.

So, geometry managers provide a **systematic, flexible, and automatic layout mechanism**.

5.1. Types of Geometry Managers

Tkinter provides **three built-in geometry managers**, each with a different approach:

Manager	Layout Style	Description
<code>pack()</code>	Relative (stacking)	Packs widgets relative to each other (top, bottom, left, right).
<code>grid()</code>	Table-like (rows & columns)	Arranges widgets in a grid layout (like a spreadsheet).
<code>place()</code>	Absolute (x, y)	Positions widgets at exact or relative coordinates.

5.1.1. `pack()` — The Packing Manager

- Packs widgets in relation to each other, **in one direction (vertical or horizontal)**.
- Works by “stacking” widgets from one side of the parent container.

When we call `widget.pack()`, Tkinter:

1. Determines the side (`TOP`, `BOTTOM`, `LEFT`, `RIGHT`).
2. Calculates how much space is required.
3. Adjusts the widget’s position and size automatically.

Common options:

```
widget.pack(  
    side=TOP,          # or LEFT, RIGHT, BOTTOM  
    fill=X,            # X, Y, BOTH  
    expand=True,        # expands to fill unused space  
    padx=10, pady=10  # adds padding around  
)
```

Example:

```
Label(root, text="Top", bg="lightblue").pack(side=TOP, fill=X)  
Label(root, text="Left", bg="lightgreen").pack(side=LEFT, fill=Y)  
Label(root, text="Right", bg="lightyellow").pack(side=RIGHT, fill=Y)
```

5.1.2. `grid()` — The Grid Manager

- Arranges widgets **in rows and columns**, similar to a table.
- Each widget is placed in a **cell** (`row`, `column`).

When you call `widget.grid()`, Tkinter:

1. Creates a table structure inside the parent.
2. Places the widget in the given row and column.
3. Expands or shrinks cells to fit the content.

Common options:

```
widget.grid(  
    row=0,
```

```

    column=1,
    rowspan=2,
    columnspan=3,
    padx=5, pady=5,
    sticky="nsew"    # aligns widget inside its cell (N, S, E, W)
)

```

Example:

```

Label(root, text="Name:").grid(row=0, column=0, sticky=E)
Entry(root).grid(row=0, column=1)
Label(root, text="Email:").grid(row=1, column=0, sticky=E)
Entry(root).grid(row=1, column=1)
Button(root, text="Submit").grid(row=2, column=1, sticky=E)

```

5.1.3. `place()` — The Placement Manager

- Positions widgets **by exact coordinates (x, y)**.
- Can also use **relative positioning** (fractions of the parent's size).

When you call `widget.place()`, Tkinter:

1. Reads `x`, `y`, or `relx`, `rely` values.
2. Positions the widget accordingly.
3. Doesn't auto-resize — layout is fixed unless relative values are used.

Common options:

```

widget.place(
    x=50, y=100,                # Absolute position
    width=100, height=30,       # Fixed size
    relx=0.5, rely=0.5,         # Relative to parent (0.0-1.0)
    relwidth=0.3, relheight=0.2,
    anchor=CENTER                # Center alignment
)

```

Example:

```

Label(root, text="Hello", bg="lightblue").place(x=50, y=50)
Button(root, text="Click Me").place(relx=0.5, rely=0.8,
    anchor=CENTER)

```

5.2. Comparison Summary

Feature	<code>pack()</code>	<code>grid()</code>	<code>place()</code>
Layout Type	Stacking	Table/Grid	Absolute
Ease of Use	Easy	Moderate	Manual
Flexibility	Low	High	Very High
Auto Resize	Yes	Yes	No (unless relative)
Mix with Others	☐	☐	☐ (but use carefully)
Best For	Simple layouts	Forms, tables	Custom UIs, games

5.3. Nested Layouts (Combination Technique)

You can use different geometry managers **in different frames**:

Example:

```
from tkinter import *

root = Tk()

frame_top = Frame(root)
frame_top.pack(fill=X)
Label(frame_top, text="Header", bg="lightblue").pack()

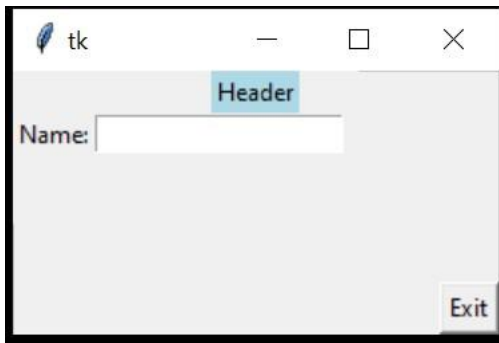
frame_main = Frame(root)
frame_main.pack(fill=BOTH, expand=True)

Label(frame_main, text="Name:").grid(row=0, column=0)
Entry(frame_main).grid(row=0, column=1)

frame_bottom = Frame(root)
frame_bottom.pack(fill=X)
Button(frame_bottom, text="Exit").pack(side=RIGHT)

root.mainloop()
```

Output:



Here, `pack()` manages the **frames**, and `grid()` manages **widgets inside frames**. This is **best practice** for complex GUIs.

5.4. Geometry Management Hierarchy

1. Each **parent widget (like Frame or root)** acts as a **container**.
2. The **geometry manager** controls the layout **within that container only**.
3. Each container can use **only one type of geometry manager**.

In Summary

- **Geometry Management** = process that decides *where and how big each widget is*.
- **pack()** → automatic stacking (easy)
- **grid()** → structured layout (most flexible)
- **place()** → manual positioning (precise but static)
- You can **combine them smartly** using Frames for clean layouts.

6. Example:

Example 1: Login Form without event-handling. **(imp)**

```

#1.import tkinter module
from tkinter import *
from tkinter import messagebox

# -----
# 2. Create object and set properties of Main window
# -----
root = Tk()
root.title("Login Form using place()")
root.geometry("400x300") # width x height
root.resizable(False, False) # disable resizing

# -----
# 3. Create Frame as container of widgets
# -----
login_frame = Frame(root, bg="lightblue", bd=2)
login_frame.place(x=50, y=50, width=300, height=200)

# -----
# 4. Create and set placement of required widgets
# -----
#Title Label
title = Label(login_frame, text="User Login", font=("Arial", 16, "bold"), bg="lightblue")
title.place(x=90, y=10) #set placement of Title Label

# -----
# Username Label and Entry
# -----
lbl_user = Label(login_frame, text="Username:", font=("Arial", 12), bg="lightblue")
lbl_user.place(x=20, y=60)

entry_user = Entry(login_frame, font=("Arial", 12))
entry_user.place(x=120, y=60, width=150)

# -----
# Password Label and Entry
# -----
lbl_pass = Label(login_frame, text="Password:", font=("Arial", 12), bg="lightblue")
lbl_pass.place(x=20, y=100)

entry_pass = Entry(login_frame, font=("Arial", 12), show="*")
entry_pass.place(x=120, y=100, width=150)

# -----
# Create Buttons widget with setting required properties
# -----
btn_login = Button(login_frame, text="Login", font=("Arial", 12, "bold"),
                    bg="green", fg="white")
btn_login.place(x=80, y=150, width=60)

btn_clear = Button(login_frame, text="Clear", font=("Arial", 12, "bold"),
                    bg="red", fg="white")
btn_clear.place(x=160, y=150, width=60)

# -----
# 5. Call mainloop() method
# -----
root.mainloop()

```

Output:



Example 2: Login Form with Event Handling

```
from tkinter import *
from tkinter import messagebox

# -----
# Main window setup
# -----
root = Tk()
root.title("Login Form using place()")
root.geometry("400x300") # width x height
root.resizable(False, False) # disable resizing

# -----
# Create Frame
# -----
login_frame = Frame(root, bg="lightblue", bd=2, relief=RIDGE)
login_frame.place(x=50, y=50, width=300, height=200)

# -----
# Title Label
# -----
title = Label(login_frame, text="User Login", font=("Arial", 16, "bold"), bg="lightblue")
title.place(x=90, y=10)

# -----
# Username Label and Entry
# -----
lbl_user = Label(login_frame, text="Username:", font=("Arial", 12), bg="lightblue")
lbl_user.place(x=20, y=60)

entry_user = Entry(login_frame, font=("Arial", 12))
entry_user.place(x=120, y=60, width=150)

# -----
# Password Label and Entry
# -----
```

```

lbl_pass = Label(login_frame, text="Password:", font=("Arial", 12), bg="lightblue")
lbl_pass.place(x=20, y=100)

entry_pass = Entry(login_frame, font=("Arial", 12), show="*")
entry_pass.place(x=120, y=100, width=150)

# -----
# Function for Login Button
# -----
def login_action():
    user = entry_user.get()
    password = entry_pass.get()

    if user == "admin" and password == "1234":
        messagebox.showinfo("Login Success", f"Welcome {user}!")
    else:
        messagebox.showerror("Login Failed", "Invalid username or password!")

# -----
# Buttons
# -----
btn_login = Button(login_frame, text="Login", font=("Arial", 12, "bold"),
                    bg="green", fg="white", command=login_action)
btn_login.place(x=80, y=150, width=60)

btn_clear = Button(login_frame, text="Clear", font=("Arial", 12, "bold"),
                    bg="red", fg="white",
                    command=lambda: [entry_user.delete(0, END), entry_pass.delete(0, END)])
btn_clear.place(x=160, y=150, width=60)

# -----
root.mainloop()

```

Example 3: Student Registration Form without event-handling.

```

# Importing required libraries
from tkinter import *          # For GUI elements
from tkinter import ttk        # For Combobox widget

# Create main window
window = Tk()
window.title("Student Registration Form")
window.geometry("500x450") # Set window size

# =====
# 1. Labels and Entry Fields
# =====
# Label for Title
Label(window, text="Student Registration
Form", font=('Arial', 20, 'bold'), fg='red').place(x=40, y=5)
# Label for Name
Label(window, text="Name:", font=('Arial', 10, 'bold')).place(x=50, y=50)
# Entry for Name
name_entry = Entry(window, width=30)
name_entry.place(x=200, y=50)

# Label for Address

```

```

Label(window, text="Address:", font=('Arial', 10, 'bold')).place(x=50, y=90)
# Entry for Address
address_entry = Entry(window, width=30)
address_entry.place(x=200, y=90)

# =====
# 2. Combobox for Program
# =====

Label(window, text="Program:", font=('Arial', 10, 'bold')).place(x=50, y=130)
program_combo = ttk.Combobox(window, values=["BCA", "BICTE", "BA"], state='readonly', width=27)
program_combo.place(x=200, y=130)
program_combo.current(0) # Default selection

# =====
# 3. Radio Buttons for Gender
# =====

Label(window, text="Gender:", font=('Arial', 10, 'bold')).place(x=50, y=170)
gender_var = StringVar(value="Male") # Default gender

Radiobutton(window, text="Male", variable=gender_var, value="Male").place(x=200, y=170)
Radiobutton(window, text="Female", variable=gender_var, value="Female").place(x=270, y=170)
Radiobutton(window, text="Other", variable=gender_var, value="Other").place(x=350, y=170)

# =====
# 4. Checkbuttons for Hobbies
# =====

Label(window, text="Hobbies:", font=('Arial', 10, 'bold')).place(x=50, y=210)

# Variables to store checkbox states
read_var = IntVar()
code_var = IntVar()
play_var = IntVar()
visit_var = IntVar()

# Creating Checkbuttons
Checkbutton(window, text="Reading", variable=read_var).place(x=200, y=210)
Checkbutton(window, text="Coding", variable=code_var).place(x=200, y=240)
Checkbutton(window, text="Playing Game", variable=play_var).place(x=200, y=270)
Checkbutton(window, text="Visit New Place", variable=visit_var).place(x=200, y=300)

# =====
# 5. Submit Button
# =====

Button(window, text="Submit", bg="lightblue", width=15).place(x=200, y=350)

# Run the main loop
window.mainloop()

```

Output:

Student Registration Form

Name:

Address:

Program:

Gender: ☒ Male ☐ Female ☐ Other

Hobbies: ☐ Reading ☐ Coding ☐ Playing Game ☐ Visit New Place

Example 4: Student Registration Form with event-handling.

```
# Importing required libraries
from tkinter import *           # For GUI elements
from tkinter import ttk        # For Combobox widget

# Create main window
window = Tk()
window.title("Student Registration Form")
window.geometry("500x450") # Set window size

# =====
# 1. Labels and Entry Fields
# =====

# Label for Name
Label(window, text="Name:", font=('Arial', 10, 'bold')).place(x=50, y=50)
# Entry for Name
name_entry = Entry(window, width=30)
name_entry.place(x=200, y=50)

# Label for Address
Label(window, text="Address:", font=('Arial', 10, 'bold')).place(x=50, y=90)
# Entry for Address
address_entry = Entry(window, width=30)
address_entry.place(x=200, y=90)

# =====
# 2. Combobox for Program
# =====
```

```

Label(window, text="Program:", font=('Arial', 10, 'bold')).place(x=50, y=130)
program_combo = ttk.Combobox(window, values=["BCA", "BICTE", "BA"], state='readonly', width=27)
program_combo.place(x=200, y=130)
program_combo.current(0) # Default selection

# =====
# 3. Radio Buttons for Gender
# =====

Label(window, text="Gender:", font=('Arial', 10, 'bold')).place(x=50, y=170)
gender_var = StringVar(value="Male") # Default gender

Radiobutton(window, text="Male", variable=gender_var, value="Male").place(x=200, y=170)
Radiobutton(window, text="Female", variable=gender_var, value="Female").place(x=270, y=170)
Radiobutton(window, text="Other", variable=gender_var, value="Other").place(x=350, y=170)

# =====
# 4. Checkbuttons for Hobbies
# =====

Label(window, text="Hobbies:", font=('Arial', 10, 'bold')).place(x=50, y=210)

# Variables to store checkbox states
read_var = IntVar()
code_var = IntVar()
play_var = IntVar()
visit_var = IntVar()

# Creating Checkbuttons
Checkbutton(window, text="Reading", variable=read_var).place(x=200, y=210)
Checkbutton(window, text="Coding", variable=code_var).place(x=200, y=240)
Checkbutton(window, text="Playing Game", variable=play_var).place(x=200, y=270)
Checkbutton(window, text="Visit New Place", variable=visit_var).place(x=200, y=300)

# =====
# 5. Submit Button
# =====

def submit_form():
    # Get all input values
    name = name_entry.get()
    address = address_entry.get()
    program = program_combo.get()
    gender = gender_var.get()

    # Collect hobbies
    hobbies = []
    if read_var.get():
        hobbies.append("Reading")
    if code_var.get():
        hobbies.append("Coding")
    if play_var.get():
        hobbies.append("Playing Game")
    if visit_var.get():
        hobbies.append("Visit New Place")

    # Display the collected info in console
    print("==== Student Registration Details ====")
    print(f"Name: {name}")
    print(f"Address: {address}")
    print(f"Program: {program}")

```

```
print(f"Gender: {gender}")
print(f"Hobbies: {' , '.join(hobbies)}")
print("=====")

# Button to submit form
Button(window, text="Submit", command=submit_form, bg="lightblue", width=15).place(x=200,
y=350)

# Run the main loop
window.mainloop()
```

Output: