

Chapter 4

Development of Graphical User Interface (GUI)

Student Learning Outcomes:

By the end of this chapter, students will be able to :

- Design interactive GUI-based programs using the Tkinter library.
- Connect Python applications to databases and perform CRUD operations.

Introduction

In this chapter, students will learn to develop user-friendly Graphical User Interfaces (GUIs) using Tkinter, Python's built-in GUI toolkit. They will create windows, frames, labels, buttons, entry fields, and manage layouts using `pack()`, `grid()`, and `place()`. They will practice event handling by linking user actions with functions and developing a simple login form. The chapter also introduces database concepts, SQL, Python database connectivity, and basic CRUD operations using SQLite or MySQL.

4.1 Graphical User Interface (GUI) Development with Tkinter

Graphical User Interface (GUI) development allows programs to interact with users through visual elements like windows, buttons, labels, and text boxes (as shown in Figure 4.1). GUIs make applications more user-friendly and easier to use compared to command-line interfaces. Python provides tools to create GUI applications efficiently. Tkinter is Python's built-in library for creating GUI programs. It is simple, widely used, and allows developers to design interactive applications quickly.

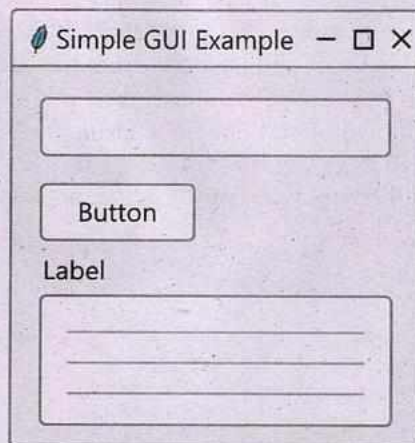


Figure 4.1: Graphical User Interface (GUI)



What is a GUI? Why is it important?

A GUI, or Graphical User Interface, is a visual interface that allows users to interact with software using elements like buttons, menus, and forms. GUIs are important because they make programs easier to understand and operate, especially for non-technical users. They improve usability by providing visual feedback and reducing the need to remember complex commands. GUI applications are widely used in desktop software, web applications, and mobile apps.

Overview of Tkinter as Python's built-in GUI toolkit

Tkinter is Python's standard GUI toolkit that comes pre-installed with the language. It provides a set of classes and functions to create windows, buttons, labels, entry fields, and other widgets. Tkinter is easy to learn, lightweight, and supports event-driven programming, where actions are triggered by user events. It is suitable for building small to medium-sized GUI applications quickly and efficiently.

Example:

Code:

```
import tkinter as tk

# Create main window
window = tk.Tk()
window.title("Simple GUI Example")
window.geometry("300x200")

# Function to handle button click (event)
def on_click():
    label.config(text="Button Clicked!")

# Create a label
label = tk.Label(window, text="Welcome to Tkinter")
label.pack(pady=10)

# Create an entry field
entry = tk.Entry(window)
entry.pack(pady=10)

# Create a button
button = tk.Button(window, text="Click Me", command=on_click)
button.pack(pady=10)

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

Output:



- Tk() → creates main window
- Label → displays text
- Entry → takes user input
- Button → performs action



- `command=on_click` → event handling (button click)
- `mainloop()` → runs the GUI

Tkinter Components and Widgets

Tkinter provides many components, called widgets, that help build interactive graphical applications. A program can contain multiple visual elements such as windows, frames, buttons, and text fields. These widgets allow users to enter data, choose options, or perform actions. Each widget has its own purpose, and combining them creates a complete interface. Tkinter also supports layout management, so widgets can be arranged neatly on the screen.



CLASS ACTIVITY

Create a main window and divide it into two frames. Add different widgets in each frame to organize layout.

Creating Windows and Adding Frames

A Tkinter program begins by creating a main window, which serves as the base for all other elements. The window can be resized, given a title, and designed to hold different widgets. Frames are sections within the window that help organize content. Frames divide the window into smaller parts, making it easier to group related widgets. This structure improves clarity and keeps the interface well-arranged.

Example:

Code:

```
import tkinter as tk

# Create main window
window = tk.Tk()
window.title("Tkinter Frames Example")
window.geometry("400x300")

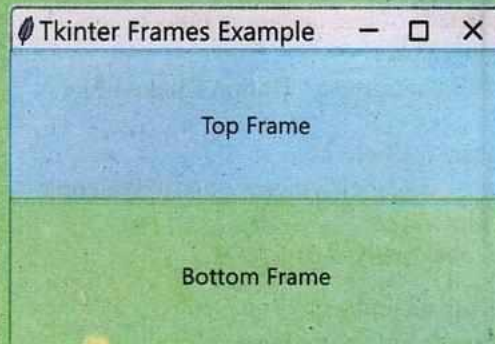
# Create top frame
top_frame = tk.Frame(window,
bg="lightblue", height=100)
top_frame.pack(fill="x")

# Create bottom frame
bottom_frame = tk.Frame(window,
bg="lightgreen", height=200)
bottom_frame.pack(fill="both",
expand=True)

# Add widgets to top frame
label_top = tk.Label(top_frame,
text="Top Frame", bg="lightblue")
label_top.pack(pady=20)

# Add widgets to bottom frame
```

Output:





```
label_bottom = tk.Label(bottom_frame,  
text="Bottom Frame", bg="lightgreen")  
label_bottom.pack(pady=50)
```

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

- Tk() → creates main window
- Frame → divides window into sections
- top_frame and bottom_frame → organize layout
- pack() → places frames and widgets
- Frames help keep the GUI clean and well-structured

Common Widgets: Labels, Buttons, Entry Fields, Menus, and List boxes

Widgets are the basic elements used to build a graphical user interface. A Label is used to display text or messages on the screen. A Button allows the user to perform an action when it is clicked. An Entry field is used to enter text, such as a name or password. A Menu provides a list of options at the top of the window. A Listbox shows a list of items and allows selection from the list.

Layout Management

Layout management is used to arrange widgets inside a window in a proper way. It decides the position and size of each widget on the screen. Good layout management makes the interface clear and easy to use. It also helps in keeping the window organized. Tkinter provides different methods to manage layout. Each method has its own way of placing widgets.



TIDBIT

Proper layout management helps make applications look neat and work well on different screen sizes.

Organizing Elements using pack(), grid(), and place()

The pack() method places widgets in a simple vertical or horizontal order. It is easy to use and works well for small applications. The grid() method arranges widgets in rows and columns like a table. It is useful for forms and structured layouts. The place() method positions widgets at exact locations on the window (as shown in Figure 4.2). It gives more control but needs careful adjustment.

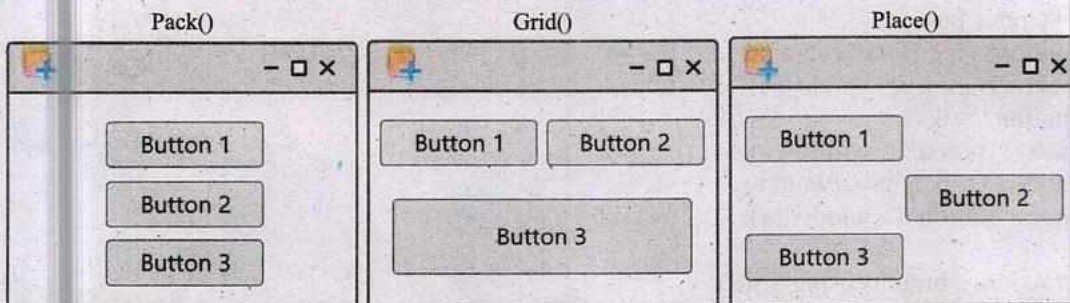


Figure 4.2: Diagram showing Tkinter Layout Managers: pack(), grid(), place()

Example pack():

Code:

```
import tkinter as tk

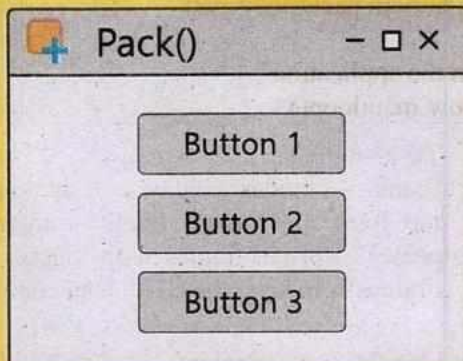
# Create main window
window = tk.Tk()
window.title("pack()")
window.geometry("300x220")

# Create buttons
button1 = tk.Button(window,
text="Button 1", width=12)
button2 = tk.Button(window,
text="Button 2", width=12)
button3 = tk.Button(window,
text="Button 3", width=12)

# Arrange buttons using pack()
button1.pack(pady=10)
button2.pack(pady=10)
button3.pack(pady=10)

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

Output:



Example grid():

Code:

```
import tkinter as tk

# Create main window
window = tk.Tk()
window.title("grid()")
window.geometry("300x220")

# Create buttons
button1 = tk.Button(window,
text="Button 1", width=10)
button2 = tk.Button(window,
text="Button 2", width=10)
button3 = tk.Button(window,
text="Button 3", width=24)

# Arrange buttons using grid()
button1.grid(row=0, column=0, padx=10,
pady=20)
```

Output:





```
button2.grid(row=0, column=1, padx=10,  
pady=20)
```

```
# Button 3 spans two columns  
button3.grid(row=1, column=0,  
columnspan=2, padx=10, pady=10)
```

```
# Keep content centred  
window.grid_columnconfigure(0,  
weight=1)  
window.grid_columnconfigure(1,  
weight=1)
```

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

Example place():

Code:

```
import tkinter as tk
```

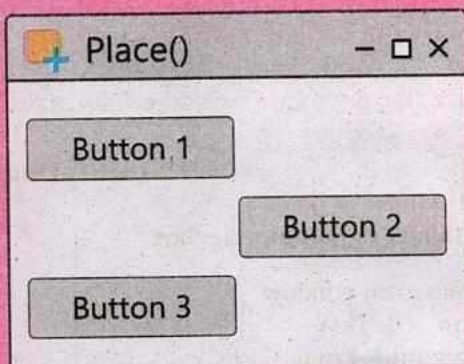
```
# Create main window  
window = tk.Tk()  
window.title("place()")  
window.geometry("300x220")
```

```
# Create buttons  
button1 = tk.Button(window,  
text="Button 1", width=10)  
button2 = tk.Button(window,  
text="Button 2", width=10)  
button3 = tk.Button(window,  
text="Button 3", width=10)
```

```
# Arrange buttons using place()  
button1.place(x=30, y=40)  
button2.place(x=170, y=85)  
button3.place(x=30, y=140)
```

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

Output:



Designing Clean and Responsive Interfaces

A clean interface has well-arranged widgets and enough space between them. Text and buttons should be clear and easy to read. Responsive interfaces adjust their layout when the window size changes. This helps the program work well on different screen sizes. Proper alignment improves the overall look of the application. Simple design makes the interface user-friendly.



Event Handling and Interactivity

Event handling and interactivity allow a program to respond to user actions. An event is an action, such as clicking a button or typing text. The program waits for an event instead of running all the time. When an event occurs, a specific task is performed. This makes the application active and useful. Event handling is important in graphical programs. It helps control how the program behaves.

Understanding Event-Driven Programming

Event-driven programming is based on actions called events. The program remains idle until an event occurs. Common events include button clicks and key presses. Each event is linked to a specific function. The function runs only when the event happens. This approach saves system resources. It is widely used in graphical user interfaces.

Handling User Input and Connecting Widgets to Functions

User input is given through widgets like entry fields and buttons. Widgets are connected to functions in the program. When the widget is used, the connected function executes. This allows the program to process the input. For example, a button click can display a message. Proper connections ensure correct program behavior. This interaction improves user experience.

Example: Simple Login Form

A simple login form allows a user to enter a username and password. Entry fields are used to take the input from the user. A button is provided to submit the information. When the button is clicked, the program checks the entered data. A message is then shown for success or error.

Example:

Code:

```
import tkinter as tk
from tkinter import messagebox

# Create main window
window = tk.Tk()
window.title("Login Form")
window.geometry("300x200")

# Function to check login
def check_login():
    username = entry_user.get()
    password = entry_pass.get()

    # simple check
    if username == "admin" and
    password == "1234":

messagebox.showinfo("Success",
"Login Successful!")
    else:
        messagebox.showerror("Error",
"Invalid Username or Password")
```

Output:





```
# Username label and entry
tk.Label(window,
text="Username").pack()
entry_user = tk.Entry(window)
entry_user.pack()

# Password label and entry
tk.Label(window,
text="Password").pack()
entry_pass = tk.Entry(window,
show="*")
entry_pass.pack()

# Login button
tk.Button(window, text="Login",
command=check_login).pack(pady=
10)

# Run application
window.mainloop()
```

- Entry → takes username and password
- Button → submits data
- check_login() → checks input
- messagebox → shows success or error
- Works on event-driven programming



CLASS ACTIVITY

Explain event-driven concepts and have students add a button that triggers a label update on click.

4.2 Working with Databases in Python

Working with databases in Python involves storing and managing data using database systems. Python programs can connect to databases to insert, retrieve, update, and delete data. Databases allow applications to handle large amounts of information efficiently. Using databases helps maintain data accuracy and consistency. Python supports database operations through built-in and external libraries.



DO YOU KNOW?

Databases allow programs to store large amounts of information safely and retrieve it quickly when needed.

Introduction to Databases

A database is an organized collection of data stored in a structured format. It allows data to be accessed and managed easily. Databases reduce data duplication and improve reliability. They



support secure storage of information. Databases are commonly used in many types of software applications.

Entity: An entity is a real-world object, person, place, event, or concept about which data is stored in a database.

Example: Student, Teacher, Course, Book, Employee

Attribute: An attribute is a property or characteristic of an entity. It describes specific information about an entity.

Example: For the entity Student, attributes can be:

- Student Name
- Roll Number
- Class
- Age

Relationship: A relationship shows how two or more entities are connected with each other.

Example: A Student enrolls in a Course. Here, enrolls in is the relationship between Student and Course.

Identifier: An identifier is an attribute that uniquely identifies each record or object of an entity.

Example: For the entity Student, Roll Number can be used as an identifier because each student has a unique roll number.

Understanding Databases

Databases store important data required for application operation. They allow applications to manage large data sets effectively. Key database concepts are:

- **Relational Database:** A database that stores data in tables (rows and columns) and links them using relationships.
- **Table:** A collection of related data organised in rows (records) and columns (fields). (as shown in Figure 4.3).
- **Column:** A property or characteristic of an entity (e.g., name, age).
- **Primary Key:** A unique identifier for each record in a table.
- **Foreign Key:** A field that links one table to another by referring to the primary key of another table.

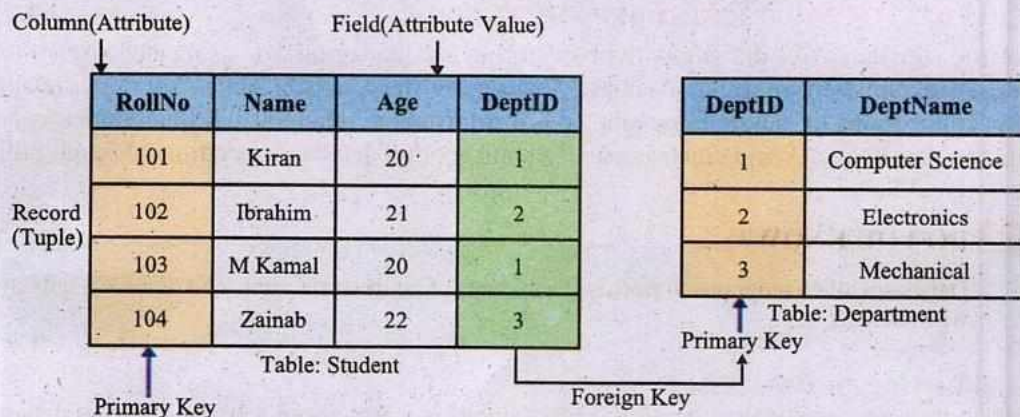


Figure 4.3: Tables, Columns, Records, Fields, Primary Key, and Foreign Key Relationship



Overview of Relational Databases and SQL Structure

Relational databases organize data into tables made of rows and columns. Each table stores related data in a structured manner. Relationships between tables are maintained using keys. SQL stands for Structured Query Language and is used to interact with databases. SQL commands help create, read, update, and delete data.

Connecting Python to a Database

Connecting Python to a database allows programs to work with stored data. A database connection is required before performing any data operation. Python provides libraries that help establish this connection easily. Once connected, data can be added, retrieved, or modified. Proper connection handling ensures smooth communication between Python and the database.

Using sqlite3 or MySQL to Establish a Database Connection

The sqlite3 library is built into Python and is used for small databases. It stores data in a single file on the system. MySQL is used for larger and more complex database systems. A connection is created by providing database details such as name and credentials. A successful connection allows Python to access the database.

```
# Import sqlite3 module
import sqlite3

# Connect to the database file named "school.db"
# If the file does not exist, SQLite will create it automatically
connection = sqlite3.connect("school.db")

# Create a cursor object
# Cursor is used to execute SQL queries
cursor = connection.cursor()

# Create "students" table if it does not exist
cursor.execute(""" CREATE TABLE IF NOT EXISTS students
( id INTEGER PRIMARY KEY, name TEXT, age INTEGER, grade TEXT ) """)

# (Optional) Insert sample data
cursor.execute ("INSERT INTO students (name, age, grade) VALUES ('Ali', 14, '8th')")
cursor.execute ("INSERT INTO students (name, age, grade) VALUES ('Sara', 15, '9th')")

# Save changes
connection.commit()

# This query selects all columns (*) from the table named "students"
cursor.execute("SELECT * FROM students")

# Fetch all records returned by the query
# fetchall() returns a list of rows/records
records = cursor.fetchall()

# Display all records
# Loop through each row in the records list
for row in records:
    # Print each row of the table
    print(row)

# Close the database connection
connection.close()
```

- **Connect to Database:** Establishes a connection to the school.db SQLite database.
- **Create Cursor:** Creates a cursor object to execute SQL commands.
- **Execute Query:** Executes a SQL query to fetch all records from the students table.
- **Fetch and Display Data:** Fetches all records and prints each row in the result.
- **Close Connection:** Closes the database connection after the operation.

Executing SQL Commands From Python

SQL commands are used to interact with the database from Python. Python sends these commands using a database cursor. Commands can be used to create tables or insert data. Data can also be retrieved using select queries. Executing SQL commands allows Python programs to manage database data effectively.

CRUD Operations

CRUD stands for Create, Read, Update, and Delete and defines basic database actions. These operations are used to manage data stored in a database (as shown in Figure 4.4 and Table 4.1). Every database-based application depends on CRUD operations. They help maintain accurate and up-to-date information. Python uses SQL commands to perform these operations on databases.

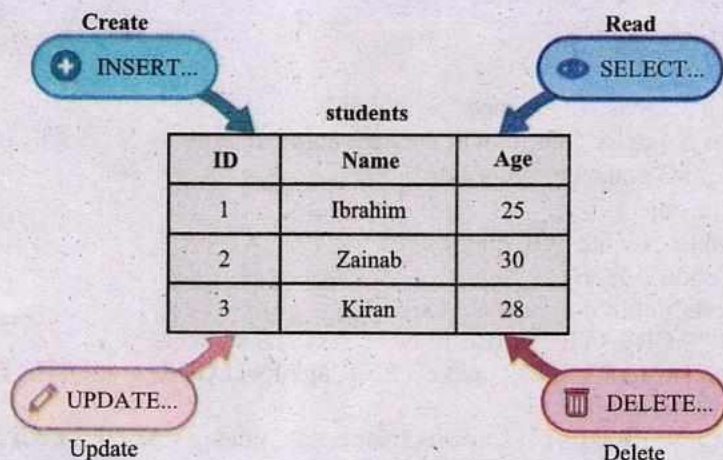


Figure 4.4: Diagram showing Create, Read, Update, and Delete operations on a table

Table 4.1: CRUD operations

Command	Command Example	Description
INSERT	INSERT INTO students (ID, Name, Age) VALUES (1, 'John', 20);	Adds a new record to a table.
READ	SELECT * FROM students;	Retrieves data from a table.
UPDATE	UPDATE students SET Age = 21 WHERE ID = 1;	Modifies an existing record in a table.
DELETE	DELETE FROM students WHERE ID = 1;	Removes a record from a table.

Create: Adding New Records

The Create operation is used to add new data to a database table. It stores information such as names, values, or records. SQL insert commands are used for this purpose. Python sends these commands to the database. New records become part of the stored data after execution.



Read: Retrieving and Displaying Data

The Read operation is used to fetch data from the database. It helps display stored records when needed. SQL select commands are commonly used for reading data. Python retrieves the results and processes them. This operation does not change the stored data.

Update: Modifying Existing Records

The Update operation changes existing data in a database. It is used when stored information needs correction. SQL update commands modify selected records. Python executes these commands based on conditions. Only the specified data is changed during this process.



CLASS ACTIVITY

Update a student's marks and display updated data.

Delete: Removing Data Safely

The Delete operation removes unwanted data from a database. It helps keep the database clean and accurate. SQL delete commands are used with conditions. Python ensures only selected records are removed. Careful use prevents loss of important data.

Example:

```
import sqlite3
# Connect to the database
connection = sqlite3.connect("school.db")
# Create a cursor object
cursor = connection.cursor()
# Execute DELETE query (example: delete student with ID = 1)
cursor.execute("DELETE FROM students WHERE ID = 1")
# Save changes
connection.commit()
# Display confirmation
print("Record deleted successfully")
# Close the connection
connection.close()
```

- DELETE FROM students WHERE ID = 1 → removes a specific record
- connection.commit() → saves the changes in database
- Without commit(), deletion will not be permanent



Summary

Graphical User Interface

A Graphical User Interface (GUI) is a visual way of interacting with a computer program using windows, buttons, menus, text boxes, and other graphical elements.

Tkinter

Tkinter is Python's built-in library used to create desktop-based graphical user interfaces.

Window

A window is the main screen or container in a GUI application where different interface elements are displayed.

Frame

A frame is a container used to organize and group widgets inside a GUI window.

Widget

A widget is a GUI element, such as a label, button, entry field, menu, or listbox, used to display information or receive input from the user.

Event-Driven Programming

Event-driven programming is a programming approach in which the program responds to events or actions performed by the user.

Relational Database

A relational database stores data in tables consisting of rows and columns, where related data can be connected using keys.

SQL

Structured Query Language (SQL) is a language used to create, manage, and manipulate data in relational databases.

Create Operation

The Create operation is used to add new records to a database.

Read Operation

The Read operation is used to retrieve and display data from a database.

Update Operation

The Update operation is used to modify existing records in a database.

Delete Operation

The Delete operation is used to remove records from a database safely.



EXERCISE

Multiple Choice Questions

1. The main purpose of a GUI in Python programming is to:
(a) Create a user-friendly interface for interacting with software
(b) Manage files in Python
(c) Handle network operations
(d) Store and retrieve data from databases
2. Python's built-in GUI toolkit is:
(a) wxPython
(b) Tkinter
(c) PyQt
(d) Kivy
3. The Tkinter widget used to display text is:
(a) Button
(b) Label
(c) Entry
(d) Listbox
4. The pack() method in Tkinter is used to:
(a) Align widgets vertically or horizontally
(b) Organize widgets in rows and columns
(c) Set widget sizes manually
(d) Create pop-up windows
5. The methods used to organize widgets in Tkinter include:
(a) grid()
(b) pack()
(c) place()
(d) All of the above
6. Event-driven programming in Tkinter means:
(a) Writing code that runs without user input
(b) Writing code that responds to user actions such as clicks or key presses
(c) Writing code to manage database queries
(d) Writing code to handle network requests
7. The Tkinter widget used to get user input is:
(a) Label
(b) Button
(c) Entry
(d) Listbox
8. The Tkinter layout manager that allows precise positioning of widgets using coordinates is:
(a) pack()
(b) grid()
(c) place()
(d) align()
9. The Tkinter option that connects a button click to a function is:
(a) bind()
(b) configure()
(c) command= (no parentheses)
(d) execute()



10. CRUD operations in databases stand for:

- (a) Create, Read, Update, Delete
- (b) Create, Run, Upload, Delete
- (c) Copy, Retrieve, Upload, Download
- (d) Create, Remove, Update, Manage



Short Questions

- 1. What is a GUI and why is it important in application development?
- 2. What is Tkinter and why is it Python's built-in GUI toolkit?
- 3. How do you create a window and add frames in Tkinter?
- 4. Name two common widgets used in Tkinter.
- 5. What is the purpose of layout management in Tkinter?
- 6. How does the pack() method organize elements in Tkinter?
- 7. What is event-driven programming and how is it used in Tkinter?
- 8. How do you handle user input in Tkinter?
- 9. What is the CRUD operation in database management?
- 10. How do you connect Python to a database like SQLite?



Long Questions

- 1. Explain the concept of Graphical User Interface (GUI) and discuss its importance in application development.
- 2. Describe Tkinter as Python's built-in GUI toolkit.
- 3. What are the key components and widgets in Tkinter?
- 4. What is layout management in Tkinter, and how are elements organized using pack(), grid(), and place()?
- 5. How can you design clean and responsive user interfaces in Tkinter?
- 6. Explain the concept of event-driven programming in Tkinter.
- 7. How do you handle user input and connect widgets to functions in Tkinter?
- 8. Discuss the process of connecting Python to a database using SQLite or MySQL.



Answer Key

- 1. A
- 2. B
- 3. B
- 4. A
- 5. D
- 6. B
- 7. C
- 8. C
- 9. C
- 10. A