**TECHNICAL TRAINING REPORT**

**BACHELOR OF TECHNOLOGY**

**COMPUTER SCIENCE AND ENGINEERING**

**YEAR-**2023 **SEM-** 4th **SECTION-** C **BATCH-2024**

****

2301320109018 Under the Guidance of

VISHU SHARMA Prof. (Dr.) Roop Ranjan

2301320109013

PRIYANKA KUMARI

**Greater Noida Institute of Technology, Greater Noida**

**Dr. A.P.J. Abdul Kalam Technical University, Lucknow**

July 2024

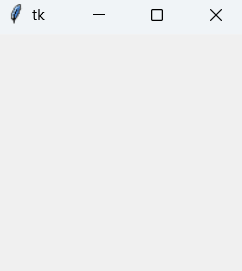
| **S.No** | **Title of the Program** | **Date** | **Signature** |
| --- | --- | --- | --- |
| 1 | To set up the main window. | 17/04/24 |  |
| 2 | To create Simple GUI app with a button using Tkinter Library. | 17/04/24 |  |
| 3 | To create a GUI window with a label widget. | 24/07/24 |  |
| 4 | To create a simple GUI with the checkbox. | 24/07/24 |  |
| 5 | To create a simple GUI with three frames. | 24/07/24 |  |
| 6 | To create a login page with a grid method. | 01/05/24 |  |
| 7 | To create a simple login using Tkinter,which is a popular GUI. | 01/05/24 |  |
| 8 | Example of CRUD app. Using Tkinter & MySql in Python. | 08/05/24 |  |
| 9 | To create class Myclass1 and demonstrate its usage. | 08/05/24 |  |
| 10 | To create class Employee and demonstrate its usage. | 15/05/24 |  |
| 11 | Demonstrating Inheritance in Python. | 15/05/24 |  |
| 12 | To define two class ,shape & rectangle , where rectangle inherits. | 15/05/24 |  |
| 13 | To create multiple inheritance in python. | 15/05/24 |  |
| 14 | Polymorphism method overloading | 29/05/24 |  |
| 15 | Method overriding & Ducking typing. | 29/05/24 |  |
| 16 | To perform addition Using instance methods. | 05/06/24 |  |

**To Setup the main window.**

import tkinter

r = tkinter.TK()

r.mainloop()



**To Create a simple GUI application with a button using the Tkinter Library.**

import tkinter as tk

r = tk.Tk()

r.title(‘Counting Seconds’)

button = tk.Button(r,text=’Stop’,width=25,bg=’red’)

button.pack()

r.mainloop()



**To create a GUI application with a button that triggers a message box when clicked.**

from tkinter import \*

from tkinter import messagebox

def helloButton():

messagebox.showinfo("Hello Python", "Hello World")

top = Tk()

top.geometry("200x200")

B = Button(top, text="Hello", command=helloButton)

B.place(x=50, y=50)

top.mainloop()

****

**To create a graphical user interface (GUI) Window with a label widget**

import tkinter as tk

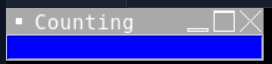
r = tk.Tk()

r.title("Counting Seconds")

entry = tk.Entry(fg="yellow", bg="blue", width=25)

entry.pack()

r.mainloop()



**To create a graphical user interface(GUI) window with a label widget it creates a window (top),add a label (L1) with the Text “User Name”, and adds an entry widget(E1) for user input.**

from tkinter import \*

top = Tk()

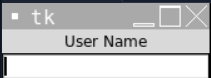
L1 = Label(top, text="User Name")

L1.pack()

E1 = Entry()

E1.pack()

top.mainloop()



**To create a simple graphical user interface(GUI) with a checkboxes. It creates a window containing two checkboxes(C! and C2) labeled “Audio” and “Video” respectively. These checkboxes allow the user to select options related to audio and video.**

from tkinter import \*

import tkinter

top = Tk()

CheckVar1 = IntVar()

CheckVar2 = IntVar()

C1 = Checkbutton(text = "Audio", variable = CheckVar1, onvalue = 1, offvalue = 0, height = 5, width = 20)

C2 = Checkbutton(text = "Video", variable = CheckVar2, onvalue = 1, offvalue = 0, height = 5, width = 20)

C1.pack()

C2.pack()

top.mainloop()



**To create a simple GUI Layout with three frames(frame1,frame2,and frame3)packed horizontally within the main window(win). Each frame has different sizes and background colors to demonstrate the layout capabilities of the pack geometry manager in Tkinter.**

import tkinter as tk

win = tk.Tk()

frame1 =tk.Frame(master=win,width=200,height=100,bg="Yellow")

frame1.pack(fill=tk.BOTH,side=tk.LEFT,expand=True)

frame2 =tk.Frame(master=win,width=100,height=100,bg="Red")

frame2.pack(fill=tk.BOTH,side=tk.LEFT,expand=True)

frame3 =tk.Frame(master=win,width=50,height=100,bg="Blue")

frame3.pack(fill=tk.BOTH,side=tk.LEFT,expand=True)

win.mainloop()

****

**To Create a Login page by using GRID method.**

from tkinter import\*

import tkinter as tk

master = tk.Tk()

L1 = Label(master,text="UserName")

L2 = Label(master,text="Password")

L1.grid(row=0,column=0,sticky=W,pady=20)

L2.grid(row=1,column=0,sticky=W,pady=2)

e1 = Entry(master)

e2 = Entry(master)

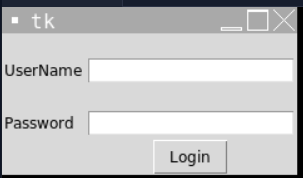
e1.grid(row=0,column=1,pady=2,padx=2)

e2.grid(row=1,column=1,pady=2,padx=2)

b1 = Button(master,text="Login")

b1.grid(row=2,column=1)

master.mainloop()



**To create a simple login form using Tkinter,which a popular GUI (Graphical User Interface) Library for Python**

from tkinter import\*

import tkinter as tk

top = tk.Tk()

top.geometry("400x250")

Username= Label(top,text="UserName").place(x=20,y=50)

email = Label(top,text="Email").place(x=20,y=90)

password = Label(top,text="Password").place(x=20,y=130)

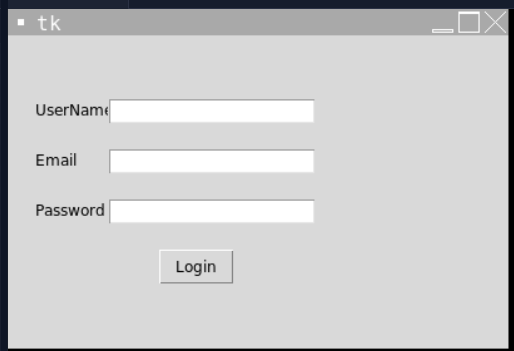
e1 = Entry(top).place(x=80,y=50)

e2 = Entry(top).place(x=80,y=90)

e3 = Entry(top).place(x=80,y=130)

b1 = Button(top,text="Login").place(x=120,y=170)

top.mainloop()

****

**This is an Example of a simple CRUD (Create,Read,Update,Delete) application using Tkinter and MySql in Python.**

from tkinter import \*

import tkinter.messagebox as MessageBox

import mysql.connector as mysql

root = Tk()

root.geometry("500x300")

root.title("MySQL CRUD Operations")

def Insert():

id = id\_entry.get()

name = name\_entry.get()

phone = phone\_entry.get()

if(id == "" or name == "" or phone == ""):

MessageBox.showinfo("ALERT", "Please enter all fields")

else:

con = mysql.connect(host="localhost", user="root", password="0402", database="EMP")

cursor = con.cursor()

cursor.execute("insert into emp\_tbl values(%s, %s, %s)", (id, name, phone))

cursor.execute("commit")

MessageBox.showinfo("Status", "Successfully Inserted")

con.close()

def Delete():

if(id\_entry.get() == ""):

MessageBox.showinfo("ALERT", "Please enter ID to delete row")

else:

con = mysql.connect(host="localhost", user="root", password="0402", database="EMP")

cursor = con.cursor()

cursor.execute("delete from emp\_tbl where id=%s", (id\_entry.get(),))

cursor.execute("commit")

MessageBox.showinfo("Status", "Successfully Deleted")

con.close()

def Select():

if(id\_entry.get() == ""):

MessageBox.showinfo("ALERT", "ID is required to select row!")

else:

con = mysql.connect(host="localhost", user="root", password="0402", database="EMP")

cursor = con.cursor()

cursor.execute("select \* from emp\_tbl where id=%s", (id\_entry.get(),))

rows = cursor.fetchall()

for row in rows:

name\_entry.insert(0, row[1])

phone\_entry.insert(0, row[2])

con.close()

def Update():

id = id\_entry.get()

name = name\_entry.get()

phone = phone\_entry.get()

if(id == "" or name == "" or phone == ""):

MessageBox.showinfo("ALERT", "Please enter all fields")

else:

con = mysql.connect(host="localhost", user="root", password="0402", database="EMP")

cursor = con.cursor()

cursor.execute("update emp\_tbl set name=%s, phone=%s where id=%s", (name, phone, id))

cursor.execute("commit")

MessageBox.showinfo("Status", "Successfully Updated")

con.close()

Label(root, text="Enter ID:", font=("verdana", 15)).place(x=50, y=30)

id\_entry = Entry(root, font=("verdana", 15))

id\_entry.place(x=200, y=30)

Label(root, text="Name:", font=("verdana", 15)).place(x=50, y=80)

name\_entry = Entry(root, font=("verdana", 15))

name\_entry.place(x=200, y=80)

Label(root, text="Phone:", font=("verdana", 15)).place(x=50, y=130)

phone\_entry = Entry(root, font=("verdana", 15))

phone\_entry.place(x=200, y=130)

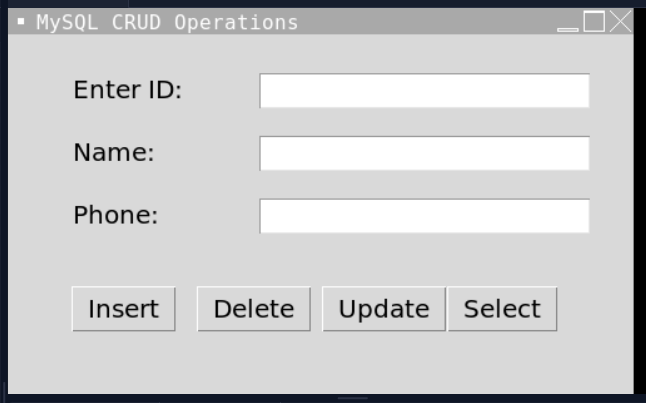
Button(root, text="Insert", command=Insert, font=("verdana", 15)).place(x=50, y=200)

Button(root, text="Delete", command=Delete, font=("verdana", 15)).place(x=150, y=200)

Button(root, text="Update", command=Update, font=("verdana", 15)).place(x=250, y=200)

Button(root, text="Select", command=Select, font=("verdana", 15)).place(x=350, y=200)

root.mainloop()



**To Create a class myclass1 and demonstrates its usage through object instantiation and method invocation.**

class myclass1:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def showdata(self):

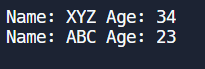
print("Name:", self.name, "Age:", self.age)

m1 = myclass1("XYZ", 34)

m2 = myclass1("ABC", 23)

m1.showdata()

m2.showdata()

****

**To create a class Employee and demonstrates its usage through object instantiation and method invocation.**

class Employee:

empCount = 0

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

Employee.empCount += 1

def print\_values(self):

print("Name:", self.name, "Age:", self.age)

m1 = Employee("XYZ", 34)

m1.print\_values()

****

**To define two classes, person and Employee, demonstrating inheritance in Python.**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display\_info(self):

print(f"Name: {self.name}, Age: {self.age}")

class Employee(Person):

def \_\_init\_\_(self, name, age, employee\_id):

super().\_\_init\_\_(name, age)

self.employee\_id = employee\_id

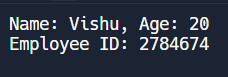
def display\_info(self):

super().display\_info()

print(f"Employee ID: {self.employee\_id}")

emp = Employee("Vishu", 20, "2784674")

emp.display\_info()

****

**To define two classes, shape and rectangle, where rectangle inherits from shape.**

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, length, breadth):

self.length = length

self.breadth = breadth

def display\_info(self):

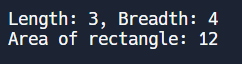
self.ar = self.length \* self.breadth

print(f"Length: {self.length}, Breadth: {self.breadth}")

print(f"Area of rectangle: {self.ar}")

rect = Rectangle(3, 4)

rect.display\_info()

****

**To create multiple inheritance in Python using classes Person, Department,and Employee**

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def display\_info(self):

print(f"Name: {self.name}, Age: {self.age}")

class Department:

def \_\_init\_\_(self, department\_name):

self.department\_name = department\_name

def display\_info(self):

print(f"Department: {self.department\_name}")

class Employee(Person, Department):

def \_\_init\_\_(self, name, age, department, employee\_id):

Person.\_\_init\_\_(self, name, age)

Department.\_\_init\_\_(self, department)

self.employee\_id = employee\_id

def display\_info(self):

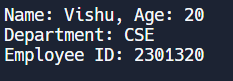
Person.display\_info(self)

Department.display\_info(self)

print(f"Employee ID: {self.employee\_id}")

emp = Employee("Vishu", 20, "CSE", "2301320")

emp.display\_info()

****

**Polymorphism Method Overloading**

class CalcSum:

def sum(self, a=None, b=None, c=None):

sum\_num = 0

if a is not None and b is not None and c is not None:

sum\_num = a + b + c

elif a is not None and b is not None:

sum\_num = a + b

elif a is not None:

sum\_num = a

return sum\_num

sum\_num = CalcSum()

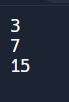
print(sum\_num.sum(3))

sum\_num = CalcSum()

print(sum\_num.sum(3, 4))

sum\_num = CalcSum()

print(sum\_num.sum(3, 5, 7))

****

**Method Overriding & Ducking typing.**

class Employee:

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

def calculate\_bonus(self):

return 0

class Manager(Employee):

def calculate\_bonus(self):

return self.salary \* 0.15

class Programmer(Employee):

def calculate\_bonus(self):

return self.salary \* 0.10

def calculate\_employee\_bonus(employee):

return employee.calculate\_bonus()

manager = Manager("Ram", 50000)

programmer = Programmer("Shyam", 60000)

print("Manager's bonus:", calculate\_employee\_bonus(manager))

print("Programmer's bonus:", calculate\_employee\_bonus(programmer))

****

**Constructor**

1. **Default Constructor**

class def\_constr:

def \_\_init\_\_(self):

self.value = "This is a default constructor"

def print\_value(self):

print(self.value)

obj = def\_constr()

obj.print\_value()

****

1. **Parameterized Constructor**

class Student:

def \_\_init\_\_(self, name):

print("This is parameterized constructor")

self.name = name

def show(self):

print("Hello", self.name)

student = Student("GNIOT")

student.show()



**Simple Python class called Addition that performs addition operations on two numbers using instance methods.**

class Addition:

def \_\_init\_\_(self, f, s):

self.first = f

self.second = s

self.answer = 0

def display(self):

print("First number = " + str(self.first))

print("Second number = " + str(self.second))

print("Addition of two numbers = " + str(self.answer))

def calculate(self):

self.answer = self.first + self.second

obj1 = Addition(1000, 2000)

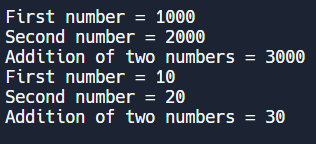
obj2 = Addition(10, 20)

obj1.calculate()

obj2.calculate()

obj1.display()

obj2.display()



**MINI PROJECT REPORT**

**TO DO LIST**

By

Vishu Sharma

(2301320109018)

&

Priyanka Kumari

(2301320109018)

Section C

CSE 2nd Year

(4th Semester)

Submitted to the department of Computer Science

For the degree of

**BACHELOR OF TECHNOLOGY**

Submitted To Prof. (Dr.) Roop Ranjan

**Overview**

Creating a to-do list application in Python with database connectivity involves designing and implementing a robust system that facilitates task management and user interaction. At its core, such an application typically includes features for user authentication, task creation, modification, deletion, categorization, and persistence of data using a database.

User Authentication: A fundamental aspect of the application is user authentication, allowing users to securely register, log in, and manage their tasks. This ensures that each user has a personalized experience and their tasks are securely stored and accessed.

Task Management: The heart of the application lies in its ability to manage tasks effectively. Users can create new tasks, provide descriptions, set priorities, assign due dates, and mark tasks as completed. These functionalities enable users to organize their daily, weekly, or long-term goals efficiently.

Database Connectivity: Integrating a database, such as SQLite or MySQL, enhances the application's functionality by providing a structured way to store and retrieve task data. The database schema typically includes tables for users and tasks, establishing relationships and ensuring data integrity. This allows for scalable and efficient data management as the application grows.

Graphical User Interface (GUI): To provide a user-friendly experience, the application utilizes a GUI framework like Tkinter or PyQt. The GUI allows users to interact with the application through intuitive buttons, input fields, dropdowns, and lists. This visual representation simplifies task management and enhances user productivity.

Functionality Expansion: Beyond basic task management, a well-rounded to-do list application may offer additional features such as:

* Search and Filter: Users can search for specific tasks or filter tasks based on criteria such as priority, due date, or status.
* Task Categories and Tags: Tasks can be categorized into different lists or tagged with keywords for better organization and quick retrieval.
* Reminders and Notifications: Automated reminders or notifications can alert users about upcoming tasks or overdue deadlines, enhancing productivity and time management.
* Data Visualization: Displaying task statistics and trends through charts or graphs can provide users with insights into their productivity and completion rates over time.
* Collaboration: Advanced applications may support collaboration features, allowing multiple users to share and work on tasks collectively, facilitating teamwork and project management.

Backend Operations: The backend operations of the application involve CRUD (Create, Read, Update, Delete) operations on both user and task data stored in the database. These operations ensure that data remains consistent, accurate, and accessible throughout the application's lifecycle.

Security Considerations: Security is paramount in handling user data and sensitive information. The application should implement secure password storage using hashing algorithms, protect against SQL injection attacks, and ensure data encryption during transmission. User authentication mechanisms, such as session management and role-based access control, also play a crucial role in safeguarding user accounts and data privacy.

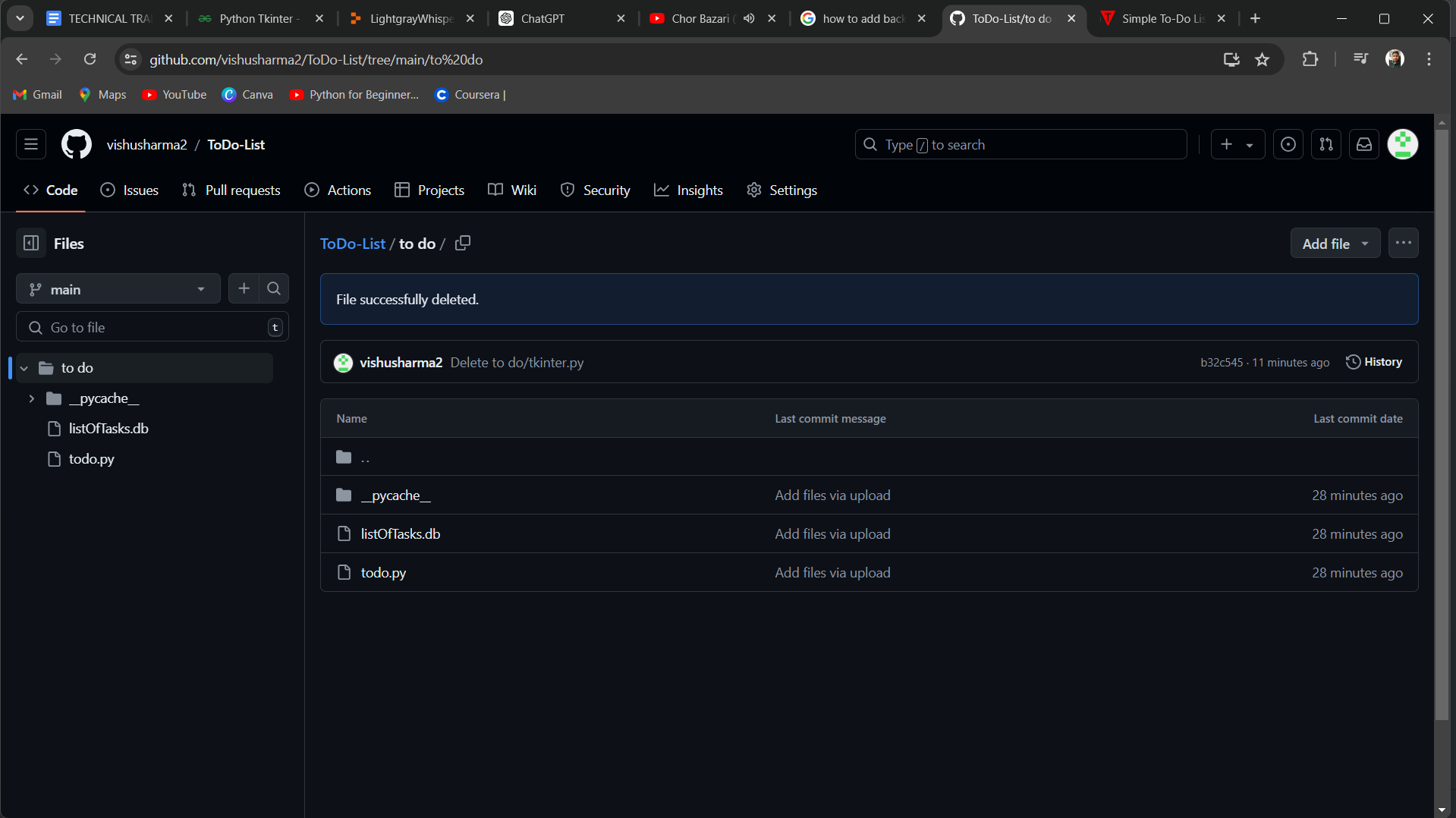
Scalability and Performance: As the application scales with more users and tasks, optimizing database queries, managing database indexes, and caching frequently accessed data become critical. Efficient code practices and database design contribute to improved application performance and responsiveness.

Learning and Development: Building a to-do list application with database connectivity in Python serves as an excellent learning exercise for understanding software development principles, database management, GUI programming, and application architecture. It allows developers to practice object-oriented programming, modular design, and debugging skills while gaining practical experience in building real-world applications.

Conclusion: In summary, developing a to-do list application in Python with database connectivity involves designing a user-friendly interface, implementing robust backend functionality, and ensuring secure and efficient data management. This project not only enhances technical proficiency but also provides valuable insights into software development practices, user experience design, and data-driven decision-making. By focusing on scalability, performance, and user security, developers can create a versatile and reliable application that meets the diverse needs of task management in personal and professional settings.

**Project on Github**

The Project can be viewed through this [link](https://github.com/vishusharma2/ToDo-List/tree/main/to%20do).



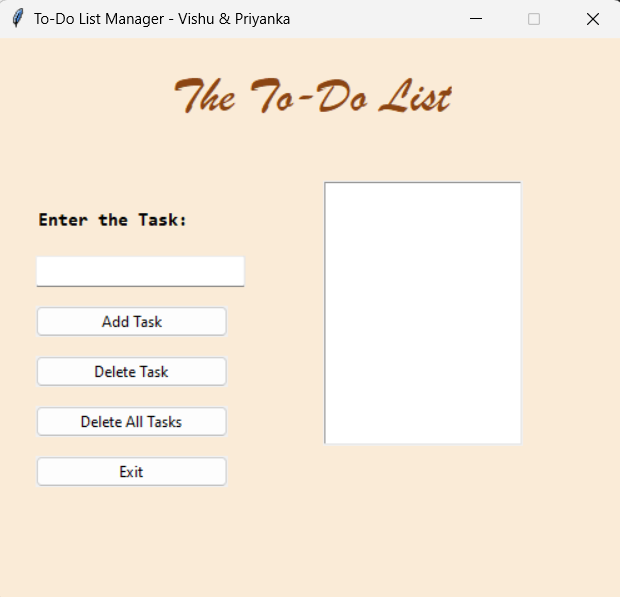
**ToDo.py**

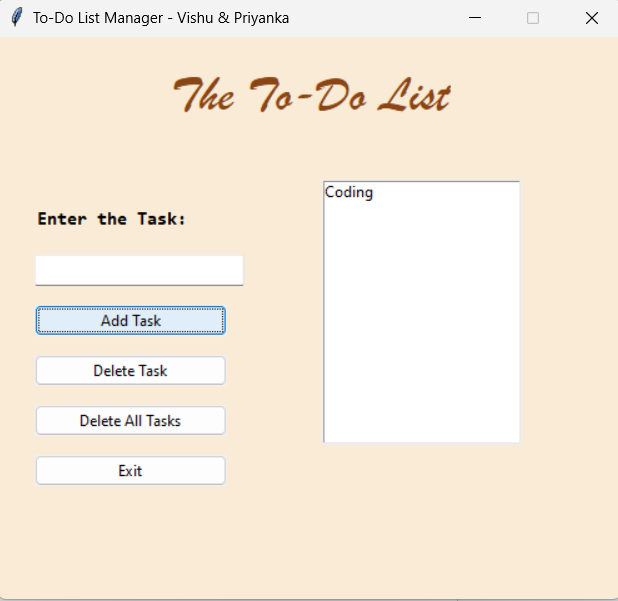
ToDo.py is a Python script that implements a comprehensive to-do list application with database connectivity. It serves as a practical example of integrating core programming concepts such as database management, graphical user interface (GUI) design, user authentication, and task management functionalities into a single cohesive application.

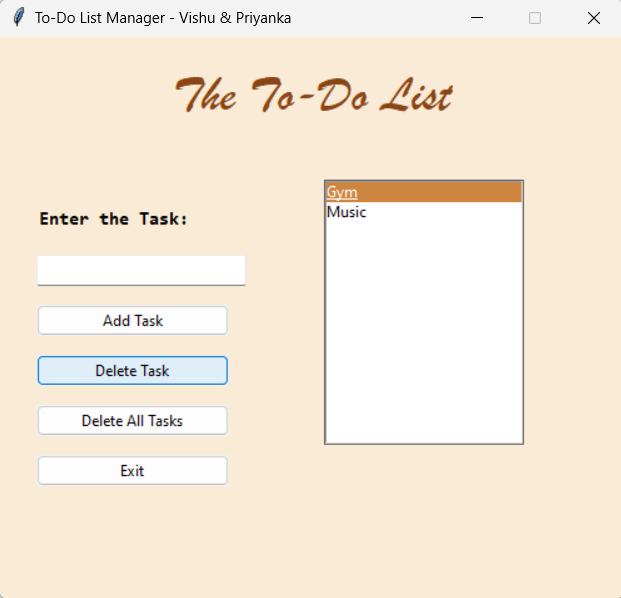
**ListOfTasks.db**

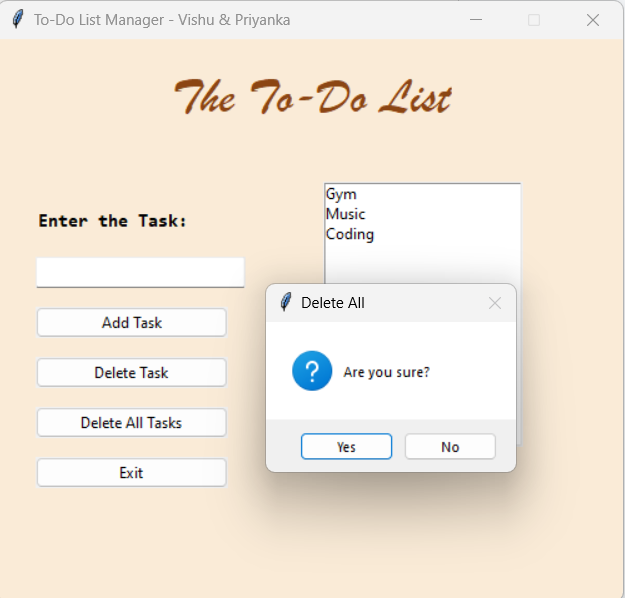
It is a database which is connected to our main file ToDo.py where we can store Add,Delete and Update our tasks which we have added in the to do list file. And can access it anytime. Here files last until we delete them.

**Screenshots**

****

****

****

****