# PIN YOUR NOTE-TECHVIDVAN

#### **About notes taker in python project:**

Notes taking is a way of keeping key information ‘noted’ down for reminders, tasks etc. This avoids us from forgetting important tasks to do, thereby keeping us updated with what has to be done. Let us see a simple implementation of the project using python.

#### **Python notes taker project:**

To implement the project, we need a database and an optional user interface. Python comes with a built-in database, SQLite3. Hence we make use of the package to store and access our notes.

#### **Project Prerequisites:**

The project requires no installation of any extra libraries. Tkinter, for the user interface and SQlite3 are built in.

If you face any error whilst importing Tkinter, refer to [Tkinter Installation](https://tkdocs.com/tutorial/install.html)

#### **Download notes taker in python Project Code:**

You can download the source code for notes taker in python in the given link: <add link>

#### **Project File Structure:**

Below is the flow of the project.

1. Importing necessary libraries
2. Creating a connection with the database and creating a table
3. Declaring functions to take, edit, view and delete notes
4. Creating a user interface

##### **Importing necessary libraries:**

#Pin Your Note -TechVidvan

#Import Necessary modules

import sqlite3 as sql

from tkinter import \*

from tkinter import messagebox

**Code Explanation:**

* **import sqlite3 as sql:** To use a database to store and retrieve our notes, we use SQlite3.
* **from tkinter import \*:** To create the user interface, we make use of Tkinter. Tkinter contains widgets and input text fields to create an interactive user interface.
* **from tkinter import messagebox:** To display the user with prompts such as information popups, warnings and question popups, we use messagebox.

##### **Creating a connection with the database and creating a table:**

# Create database connection and connect to table

try:

con = sql.connect('pin\_your\_note.db')

cur = con.cursor()

cur.execute('''CREATE TABLE notes\_table

(date text, notes\_title text, notes text)''')

except:

print("Connected to table of database")

**Code explanation:**

* **con = sql.connect('pin\_your\_note.db'):** Create a database to contain our table(s). Connect function connects to a database. It creates one if the database does not exist.
* **cur = con.cursor():** Cursor is an object that we can use to implement various functions such as execution of a query or fetching of results and so on.
* **cur.execute():** Execute executes or performs the SQL statement given to it. Here we make it create a table, “Notes\_table” with 3 columns date, notes\_title and notes each of the type text. Text data type accepts numbers, symbols and letters
* **try...except:** We put the database connection and the table creation in the try...except statement to use the table if it exists or create it.

##### **Declaring functions to take, edit, view and delete notes:**

# Insert a row of data

def add\_notes():

#Get input values

today = date\_entry.get()

notes\_title = notes\_title\_entry.get()

notes = notes\_entry.get("1.0", "end-1c")

#Raise a prompt for missing values

if (len(today) <=0) & (len(notes\_title)<=0) & (len(notes)<=1):

messagebox.showerror(message = "ENTER REQUIRED DETAILS" )

else:

#Insert into the table

cur.execute("INSERT INTO notes\_table VALUES ('%s','%s','%s')" %(today, notes\_title, notes))

messagebox.showinfo(message="Note added")

#Commit to preserve the changes

con.commit()

**Code Explanation:**

* **def add\_notes():** Declaration of the function to add notes to the database
* **today, notes\_title, notes:** Obtain the input provided by the user using get() function. For entry widget, get() does not contain any parameters, whereas we need to specify the index for a text widget (1.0 to end-1c)
* **if...else:** Check if the user provided all the inputs or raise an error using messagebox.showerror() where the message parameter
* +r indicates the message to display to the user. If all the inputs are present, then execute the SQL query to insert the inputs into the table. Use formatting operator %s to replace inputs in the required fields
* **con.commit():**To preserve the changes in the database (keep it permanent), use commit.

#Display all the notes

def view\_notes():

#Obtain all the user input

date = date\_entry.get()

notes\_title = notes\_title\_entry.get()

#If no input is given, retrieve all notes

if (len(date) <=0) & (len(notes\_title)<=0):

sql\_statement = "SELECT \* FROM notes\_table"

#Retrieve notes matching a title

elif (len(date) <=0) & (len(notes\_title)>0):

sql\_statement = "SELECT \* FROM notes\_table where notes\_title ='%s'" %notes\_title

#Retrieve notes matching a date

elif (len(date) >0) & (len(notes\_title)<=0):

sql\_statement = "SELECT \* FROM notes\_table where date ='%s'"%date

#Retrieve notes matching the date and title

else:

sql\_statement = "SELECT \* FROM notes\_table where date ='%s' and notes\_title ='%s'" %(date, notes\_title)

#Execute the query

cur.execute(sql\_statement)

#Obtain all the contents of the query

row = cur.fetchall()

#Check if none was retrieved

if len(row)<=0:

messagebox.showerror(message="No note found")

else:

#Print the notes

for i in row:

messagebox.showinfo(message="Date: "+i[0]+"\nTitle: "+i[1]+"\nNotes: "+i[2])

**Code Explanation:**

* **def view\_notes():** Declaration of the function to view notes in the database
* **date, notes\_title:** Read the user input given in the entry widgets
* **(len(date) <=0) & (len(notes\_title)<=0):** If the user does not give any input, then display all the notes from the table using the SQL query: Select \* from notes\_table
* **(len(date) <=0) & (len(notes\_title)>0):** Retrieve notes containing the given title by adding a where statement in the SQL query. The where statement is a test condition that selects queries or rows containing the title
* **(len(date) >0) & (len(notes\_title) <=0):** Retrieve notes containing the given title by adding a where statement in the SQL query. Here the where statement selects queries or rows containing the date
* **else:** Retrieve queries containing the given date and the title. To add two or more test conditions, we use AND or OR. AND returns results that satisfy test condition 1 and the other N test conditions, whereas OR returns results that satisfy at least 1 test condition of the N conditions.
* **cur.execute(sql\_statement):** Execute the select statement that passes through the conditions
* **row = cur.fetchall():** Fetchall is analogous to select all and it returns a list containing all the queries matching the selection query in a tuple data structure.
* **if len(row)<=0:** Check if the query contains any result to display. If there is none, prompt an error, else display the result by looping through the list of tuples.

#Delete the notes

def delete\_notes():

#Obtain input values

date = date\_entry.get()

notes\_title = notes\_title\_entry.get()

#Ask if user wants to delete all notes

choice = messagebox.askquestion(message="Do you want to delete all notes?")

#If yes is selected, delete all

if choice == 'yes':

sql\_statement = "DELETE FROM notes\_table"

else:

#Delete notes matching a particular date and title

if (len(date) <=0) & (len(notes\_title)<=0):

#Raise error for no inputs

messagebox.showerror(message = "ENTER REQUIRED DETAILS" )

return

else:

sql\_statement = "DELETE FROM notes\_table where date ='%s' and notes\_title ='%s'" %(date, notes\_title)

#Execute the query

cur.execute(sql\_statement)

messagebox.showinfo(message="Note(s) Deleted")

con.commit()

**Code explanation:**

* **def delete\_notes():** Declaration of the function delete\_notes() to delete notes
* **date, notes\_title:** Read user input for date and title of the note
* **choice:** Ask the user if he wants to delete all notes or only certain notes. If the user selects yes, then delete all notes. Otherwise check if the user gave the date and title of the note to delete and execute the query to delete the note. Commit the changes to the database

#Update the notes

def update\_notes():

#Obtain user input

today = date\_entry.get()

notes\_title = notes\_title\_entry.get()

notes = notes\_entry.get("1.0", "end-1c")

#Check if input is given by the user

if (len(today) <=0) & (len(notes\_title)<=0) & (len(notes)<=1):

messagebox.showerror(message = "ENTER REQUIRED DETAILS" )

#update the note

else:

sql\_statement = "UPDATE notes\_table SET notes = '%s' where date ='%s' and notes\_title ='%s'" %(notes, today, notes\_title)

cur.execute(sql\_statement)

messagebox.showinfo(message="Note Updated")

con.commit()

**Code explanation:**

* **def update\_notes():** Declaration of the function update\_notes() to update notes
* **today, notes\_title, notes:** Read user input for date, title of the note and the note
* **(len(today) <=0) & (len(notes\_title)<=0) & (len(notes)<=1):** Check if the user gave all the inputs. Raise an error, if not or execute the update statement to update the notes. Commit the changes to the database

##### **Creating a user interface:**

#Invoke call to class to view a window

window = Tk()

#Set dimensions of window and title

window.geometry("500x300")

window.title("Pin Your Note -TechVidvan")

title\_label = Label(window, text="Pin Your Note -TechVidvan").pack()

#Read inputs

#Date input

date\_label = Label(window, text="Date:").place(x=10,y=20)

date\_entry = Entry(window, width=20)

date\_entry.place(x=50,y=20)

#Notes Title input

notes\_title\_label = Label(window, text="Notes title:").place(x=10,y=50)

notes\_title\_entry = Entry(window, width=30)

notes\_title\_entry.place(x=80,y=50)

#Notes input

notes\_label = Label(window, text="Notes:").place(x=10,y=90)

notes\_entry = Text(window, width=50,height=5)

notes\_entry.place(x=60,y=90)

#Perform notes functions

button1 = Button(window,text='Add Notes', bg = 'Turquoise',fg='Red',command=add\_notes).place(x=10,y=190)

button2 = Button(window,text='View Notes', bg = 'Turquoise',fg='Red',command=view\_notes).place(x=110,y=190)

button3 = Button(window,text='Delete Notes', bg = 'Turquoise',fg='Red',command=delete\_notes).place(x=210,y=190)

button4 = Button(window,text='Update Notes', bg = 'Turquoise',fg='Red',command=update\_notes).place(x=320,y=190)

#close the app

window.mainloop()

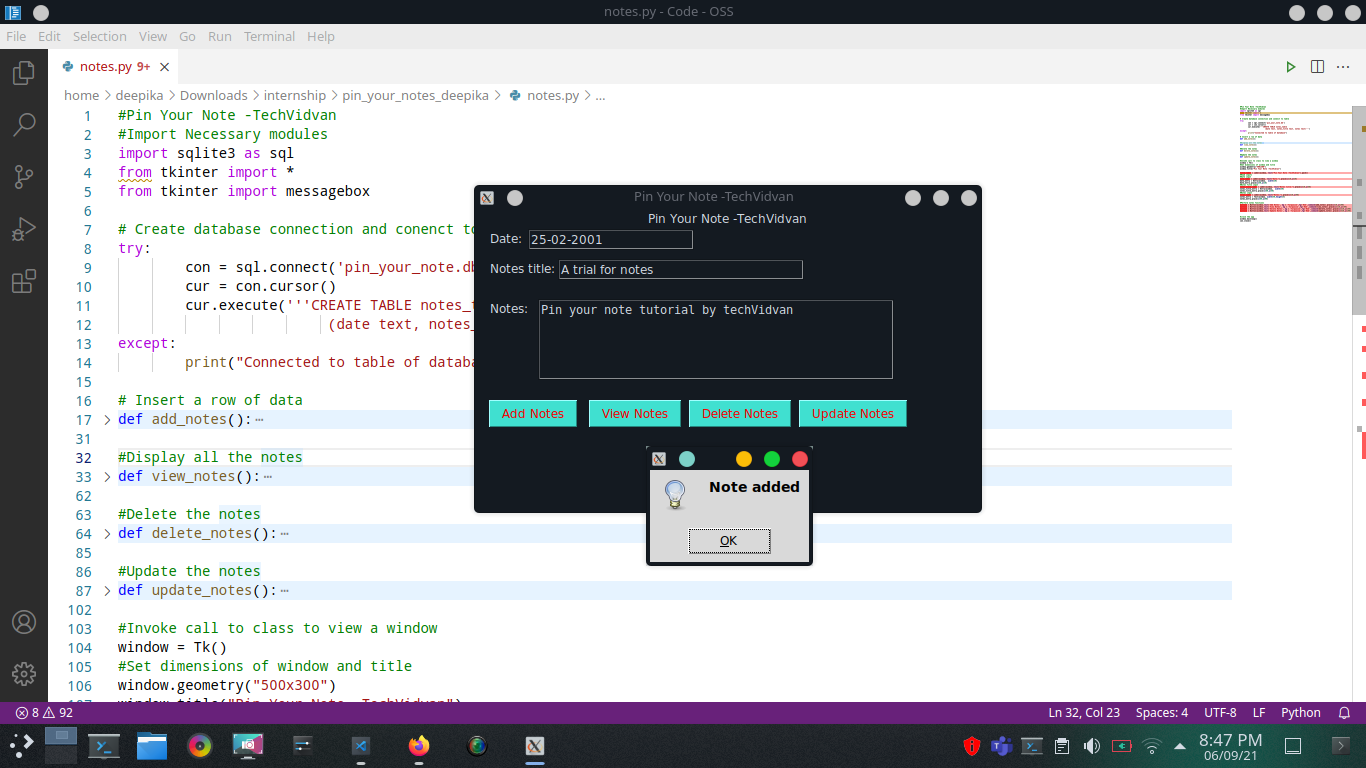
con.close()

**Code explanation:**

* **window = Tk():** Initialise the window with tkinter constructor to use the objects and widgets
* **window.geometry("500x300"):**Set the dimensions of the window by specifying the dimensions of the width and the height of the window.
* **window.title():** Add a title to the window using the title parameter
* **title\_label:** Title label indicates the title of the application. It displays a text that cannot be copied or edited. Position the element on the window to make it visible. Here we use a pack, which centers the element in the first row. So the widget remains centered regardless of the window size
* **date\_label, notes\_title\_label, notes\_label:** Define a label with the parameters: window of the screen and the text to display.
* **date\_entry, notes\_title\_entry:** Entry widget is an input field to obtain user input. Specify the width of the widget using the width parameter.
* **notes\_entry:** Text widget is another input field to obtain user input. Use this widget to obtain long lines of text from the user. Specify the height and width of the text box.
* **place():** Place is another positioning element analogous to pack(). Here we specify the distance from the left margin and the top margin in the x and y coordinates respectively.
* **button1, button2, button3, button4**: Buttons perform a function when the user selects it. The parameters are window of the application, name of the button, background colour of the application, text colour using the foreground and the function call is invoked using command parameter.
* **window.mainloop():** When the user terminates the application, the control flows beyond this line thereby terminating the application. Widgets placed after this line will not be displayed
* **con.close():** Close the connection with the database

#### **Project output:**

Run the program and view your output



#### **Summary**

Thus we found a way to implement a notes taker in python. This project is an introduction to querying using SQLite and creation of a user interface using Tkinter.