Saveetha Institute of Medical and Technical Sciences

Saveetha Scholl of Engineering, Thandalam, Chennai – 602 105.



Capstone Project

SIMPLE TO\_DO LIST APPLICATION BY USING PYTHON PROGRAMMING

Done by:-

Name: V. Tamizhselvan,

Register. No: 192224029

Guide\_Name: Dr. E.K.Subramanian

Subject: CSA0883 – Python programming for Data Visualiation

**TABLE OF CONTENTS:-**

|  |  |  |
| --- | --- | --- |
| S.NO | CONTENTS | PAGE.NO |
| 1). | Abstract | 3 |
| 2). | Introduction | 3 |
| 3). | Implementation of a Simple Todo List Application in Python | 4 |
| 4). | Advantages of a Simple To-Do List Application in Python | 4 |
| 5). | Disadvantages of a Simple To-Do List Application in Python | 5 |
| 6). | Key functionalities | 5 |
| 7). | Code Sniffets | 6 |
| 8). | Output | 15 |
| 9). | Future scopes of a Simple to-do list application developed using Python | 15 |
| 10). | Conclusion | 17 |
| 11). | Bibliography | 18 |

1). Abstract:-

This project involves developing a simple yet functional Todo List application using Python programming. The application aims to provide users with an intuitive interface to manage their daily tasks effectively. By leveraging Python's versatile libraries and modules, the application offers features such as task creation, deletion, and categorization, along with the ability to set deadlines and priorities. The design of the Todo List application focuses on ease of use and efficiency. It employs a graphical user interface (GUI) built with Tkinter, a standard Python library for creating GUI applications. Users can add new tasks, mark tasks as completed, delete tasks, and filter tasks based on their status or priority. The application also integrates with a local database (using SQLite) to persistently store user data, ensuring that task information is not lost between sessions.  By developing this Todo List application, the project aims to enhance users' productivity and organization skills, providing a practical tool for everyday task management. Additionally, it demonstrates the capability of Python in developing desktop applications, highlighting the language's simplicity and effectiveness in handling common software development tasks.

2). Introduction:-

In today's fast-paced world, staying organized and managing tasks efficiently is crucial. A simple Todo list application serves as an essential tool for individuals to keep track of their tasks, set priorities, and enhance productivity. This introduction aims to elaborate on the development of a basic Todo list application using Python, a versatile and powerful programming language. Python's simplicity and readability make it an excellent choice for beginners and experienced developers alike. It provides a wide range of libraries and frameworks that facilitate rapid application development. For this Todo list application, we will leverage Python's fundamental data structures and functionalities to create a user-friendly and efficient task management system. Developing a simple Todo list application in Python is an excellent way to enhance your programming skills while creating a practical tool for everyday use. This project will provide hands-on experience with Python's core features and demonstrate the power and flexibility of the language. Whether you are a beginner looking to learn Python or an experienced developer seeking to build a useful application, this project is an ideal starting point.

3). Implementation of a Simple Todo List Application in Python:-

**Requirements:-**

1. Python 3.x

2. Basic knowledge of file handling

3. Understanding of lists and dictionaries

**Steps:-**

**1. Define the Data Structure:** Use a list to store tasks, where each task is a dictionary containing details like the task name, due date, and status.

**2. Functions:**-

* Add Task: Function to add a new task to the list.
* View Tasks: Function to display all tasks.
* Remove Task: Function to remove a task by its index.
* Mark Task as Complete: Function to mark a task as complete.
* Save Tasks: Function to save the current list of tasks to a file.
* Load Tasks: Function to load tasks from a file.

4). Advantages of a Simple To-Do List Application in Python:-

1. Ease of Learning and Use:-

* Python's syntax is clear and easy to understand, making it accessible for beginners.
* Rapid development and prototyping due to its high-level nature.

2. Wide Range of Libraries: Availability of libraries like Tkinter for GUI, SQLite for database management, and Flask or Django for web applications.

3. Cross-Platform: Python applications can run on various operating systems (Windows, MacOS, Linux) without significant changes.

4. Community Support: Extensive community support with numerous tutorials, forums, and documentation available.

5. Flexibility: Python allows integration with other languages and technologies, making it versatile for various types of applications.

6. Readable Code: Python emphasizes readability, which makes maintaining and updating code easier.

5). Disadvantages of a Simple To-Do List Application in Python:-

1. Performance Limitations: Python can be slower compared to compiled languages like C++ or Java, which might be a concern for more complex applications.

2. Mobile Development: Python is not as popular for mobile app development compared to Swift for iOS or Kotlin for Android.

3. Global Interpreter Lock (GIL): Python's GIL can be a bottleneck in multi-threaded applications, potentially affecting performance in a to-do list app with concurrent tasks.

4. Dependency Management: Managing dependencies and ensuring compatibility across different environments can be challenging.

5. Runtime Errors: Python is dynamically typed, which means some errors might only be caught at runtime rather than at compile time.

6. Scalability: For very large applications, Python may face scalability issues, both in terms of performance and structure.

6). Key functionalities:-

1. Task Management: Users can create, update, and delete tasks. Each task can have a description, due date, and priority level.

2. Task Categorization: Tasks can be categorized into different groups, allowing users to organize their tasks based on projects or personal preference.

3. Search and Filter: Users can search for tasks and filter them based on criteria such as due date, priority, or completion status.

4. Data Persistence: The application uses SQLite to store tasks, ensuring data is saved between sessions and can be retrieved anytime.

5. User-Friendly Interface: The application provides an intuitive and visually appealing interface, making it easy for users of all ages to interact with.

**Python is chosen for this project due to its:-**

* Ease of Use: Python's syntax is clear and concise, making it easy to write and understand code.
* Extensive Libraries: Python has numerous libraries that can simplify tasks such as file handling, data manipulation, and user interface design.
* Community Support: Python has a large and active community, providing a wealth of resources, tutorials, and support.

 7). Code Sniffets:-

**The implementation of the Todo list application will involve the following steps:-**

1. Setting Up the Environment: Install Python and any necessary libraries.

2. Designing the Data Structures: Use lists and dictionaries to store tasks and their attributes.

3. Implementing Core Features: Write functions to add, delete, and view tasks.

4. File Handling: Implement file operations to save and load tasks.

5. User Interface: Develop a simple command-line interface for user interaction.

**Python Code for Simple To-Do List Application:-**

# importing the required modules

import tkinter as tk                    # importing the tkinter module as tk

from tkinter import ttk                 # importing the ttk module from the tkinter library

from tkinter import messagebox          # importing the messagebox module from the tkinter library

import sqlite3 as sql                   # importing the sqlite3 module as sql

# defining the function to add tasks to the list

def add\_task():

    # getting the string from the entry field

    task\_string = task\_field.get()

    # checking whether the string is empty or not

    if len(task\_string) == 0:

        # displaying a message box with 'Empty Field' message

        messagebox.showinfo('Error', 'Field is Empty.')

    else:

        # adding the string to the tasks list

        tasks.append(task\_string)

        # using the execute() method to execute a SQL statement

        the\_cursor.execute('insert into tasks values (?)', (task\_string ,))

        # calling the function to update the list

        list\_update()

        # deleting the entry in the entry field

        task\_field.delete(0, 'end')

  # defining the function to update the list

def list\_update():

    # calling the function to clear the list

    clear\_list()

    # iterating through the strings in the list

    for task in tasks:

        # using the insert() method to insert the tasks in the list box

        task\_listbox.insert('end', task)

# defining the function to delete a task from the list

def delete\_task():

    # using the try-except method

    try:

        # getting the selected entry from the list box

        the\_value = task\_listbox.get(task\_listbox.curselection())

        # checking if the stored value is present in the tasks list

        if the\_value in tasks:

            # removing the task from the list

            tasks.remove(the\_value)

            # calling the function to update the list

            list\_update()

            # using the execute() method to execute a SQL statement

            the\_cursor.execute('delete from tasks where title = ?', (the\_value,))

    except:

        # displaying the message box with 'No Item Selected' message for an exception

        messagebox.showinfo('Error', 'No Task Selected. Cannot Delete.')

  # function to delete all tasks from the list

def delete\_all\_tasks():

    # displaying a message box to ask user for confirmation

    message\_box = messagebox.askyesno('Delete All', 'Are you sure?')

    # if the value turns to be True

    if message\_box == True:

        # using while loop to iterate through the tasks list until it's empty

        while(len(tasks) != 0):

# using the pop() method to pop out the elements from the list

            tasks.pop()

        # using the execute() method to execute a SQL statement

        the\_cursor.execute('delete from tasks')

        # calling the function to update the list

        list\_update()

  # function to clear the list

def clear\_list():

    # using the delete method to delete all entries from the list box

    task\_listbox.delete(0, 'end')

# function to close the application

def close():

    # printing the elements from the tasks list

    print(tasks)

    # using the destroy() method to close the application

    guiWindow.destroy()

# function to retrieve data from the database

def retrieve\_database():

    # using the while loop to iterate through the elements in the tasks list

    while(len(tasks) != 0):

        # using the pop() method to pop out the elements from the list

        tasks.pop()

    # iterating through the rows in the database table

    for row in the\_cursor.execute('select title from tasks'):

        # using the append() method to insert the titles from the table in the list

tasks.append(row[0])

# main function

if \_\_name\_\_ == "\_\_main\_\_":

    # creating an object of the Tk() class

    guiWindow = tk.Tk()

    # setting the title of the window

    guiWindow.title("Simple To-Do List Application by V.Tamizhselvan")

    # setting the geometry of the window

    guiWindow.geometry("500x450+750+250")

    # disabling the resizable option

    guiWindow.resizable(0, 0)

    # setting the background color to #FAEBD7

    guiWindow.configure(bg = "#FAEBD7")

  # using the connect() method to connect to the database

    the\_connection = sql.connect('listOfTasks.db')

    # creating the cursor object of the cursor class

    the\_cursor = the\_connection.cursor()

    # using the execute() method to execute a SQL statement

    the\_cursor.execute('create table if not exists tasks (title text)')

  # defining an empty list

    tasks = []

  # defining frames using the tk.Frame() widget

    header\_frame = tk.Frame(guiWindow, bg = "#FAEBD7")

    functions\_frame = tk.Frame(guiWindow, bg = "#FAEBD7")

    listbox\_frame = tk.Frame(guiWindow, bg = "#FAEBD7")

    # using the pack() method to place the frames in the application

    header\_frame.pack(fill = "both")

    functions\_frame.pack(side = "left", expand = True, fill = "both")

    listbox\_frame.pack(side = "right", expand = True, fill = "both")

    # defining a label using the ttk.Label() widget

    header\_label = ttk.Label(

        header\_frame,

        text = "The To-Do List",

        font = ("Brush Script MT", "30"),

        background = "#FAEBD7",

        foreground = "#8B4513"

    )

    # using the pack() method to place the label in the application

    header\_label.pack(padx = 20, pady = 20)

  # defining another label using the ttk.Label() widget

    task\_label = ttk.Label(

        functions\_frame,

        text = "Enter the Task:",

        font = ("Consolas", "11", "bold"),

        background = "#FAEBD7",

        foreground = "#000000"

    )

    # using the place() method to place the label in the application

    task\_label.place(x = 30, y = 40)

    # defining an entry field using the ttk.Entry() widget

    task\_field = ttk.Entry(

        functions\_frame,

        font = ("Consolas", "12"),

        width = 18,

        background = "#FFF8DC",

        foreground = "#A52A2A"

    )

    # using the place() method to place the entry field in the application

    task\_field.place(x = 30, y = 80)

    # adding buttons to the application using the ttk.Button() widget

    add\_button = ttk.Button(

        functions\_frame,

        text = "Add Task",

        width = 24,

        command = add\_task

    )

    del\_button = ttk.Button(

        functions\_frame,

        text = "Delete Task",

        width = 24,

        command = delete\_task

    )

    del\_all\_button = ttk.Button(

        functions\_frame,

        text = "Delete All Tasks",

        width = 24,

        command = delete\_all\_tasks

    )

    exit\_button = ttk.Button(

        functions\_frame,

        text = "Exit",

        width = 24,

        command = close

    )

    # using the place() method to set the position of the buttons in the application

    add\_button.place(x = 30, y = 120)

    del\_button.place(x = 30, y = 160)

    del\_all\_button.place(x = 30, y = 200)

    exit\_button.place(x = 30, y = 240)

  # defining a list box using the tk.Listbox() widget

    task\_listbox = tk.Listbox(

        listbox\_frame,

        width = 26,

        height = 13,

        selectmode = 'SINGLE',

        background = "#FFFFFF",

        foreground = "#000000",

        selectbackground = "#CD853F",

selectforeground = "#FFFFFF"

    )

    # using the place() method to place the list box in the application

    task\_listbox.place(x = 10, y = 20)

    # calling some functions

    retrieve\_database()

    list\_update()

    # using the mainloop() method to run the application

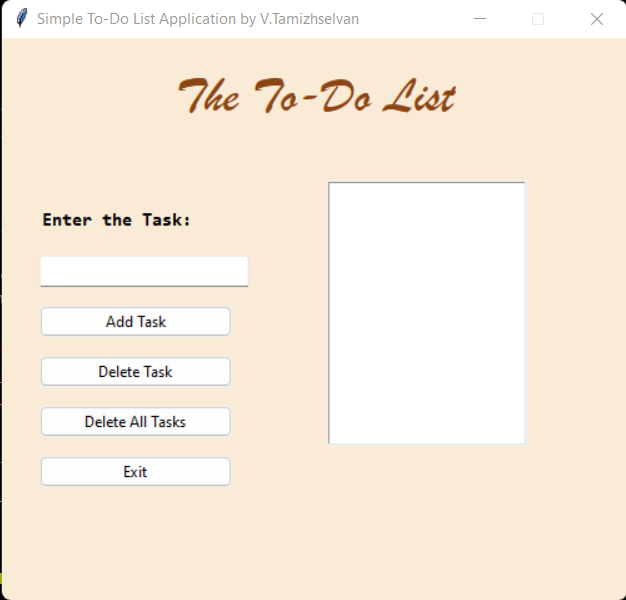
    guiWindow.mainloop()

    # establishing the connection with database

    the\_connection.commit()

    the\_cursor.close()

8). Output:-



9). Future scopes of a Simple to-do list application developed using Python:-

1. User Authentication and Profiles:-

* Implement user registration and login functionality.
* Allow users to create personalized profiles to manage their tasks.
* Store user data securely using databases like SQLite, PostgreSQL, or MongoDB.

2. Cloud Synchronization:-

* Integrate cloud services to sync tasks across multiple devices.
* Use APIs from services like Google Drive, Dropbox, or custom cloud storage solutions.

3. Collaboration Features:-

* Enable sharing of to-do lists with other users.
* Implement real-time collaboration, allowing multiple users to edit the same list simultaneously.

4. Advanced Task Management:-

* Add features for task categorization, tagging, and prioritization.
* Implement task dependencies, subtasks, and recurring tasks.
* Allow for reminders and notifications through email or push notifications.

5. Natural Language Processing (NLP):-

* Use NLP to allow users to add tasks using natural language (e.g., "Meet John for lunch at 2 PM tomorrow").
* Implement voice recognition for task input using libraries like SpeechRecognition.

 6. Integration with Other Tools:-

* Integrate with calendars (Google Calendar, Outlook) to auto-schedule tasks.
* Connect with productivity tools like Trello, Slack, or Microsoft Teams for better task management and communication.

 7. AI and Machine Learning:-

* Use machine learning algorithms to predict task completion times and suggest optimal schedules.
* Implement sentiment analysis to prioritize tasks based on user feedback or mood.

 8. Mobile and Web Applications:-

* Develop mobile applications for iOS and Android using frameworks like Kivy or BeeWare.
* Create a web-based interface using Flask or Django for remote access to the to-do list.

9. Data Analytics:-

* Provide users with insights and analytics on their productivity.
* Visualize task completion trends, time management, and other metrics using libraries like Matplotlib or Plotly.

10. Customization and Themes:-

* Allow users to customize the UI with different themes and layouts.
* Provide options for task sorting, filtering, and searching.

10). Conclusion:-

The development of a simple Todo List application using Python has demonstrated the practicality and efficiency of using a high-level programming language to manage everyday tasks. This project underscores several important aspects of software development, including:

1. User-Friendly Design: By focusing on a clear and intuitive user interface, the application ensures that users can easily add, view, update, and delete tasks. This user-centric design approach highlights the importance of accessibility and ease of use in software development.

2. Fundamental Programming Concepts: The project incorporates essential programming concepts such as data structures (lists and dictionaries), control flow (loops and conditionals), and basic file handling. These foundational skills are critical for any aspiring programmer and provide a strong base for more complex projects.

3. Problem-Solving Skills: Developing the Todo List application involved identifying user needs, defining requirements, and implementing solutions. This process enhances problem-solving skills and the ability to think logically and critically, which are vital competencies in the field of software development.

4. Code Modularity and Reusability: By organizing the code into functions and modules, the project promotes modularity and reusability. This practice not only makes the code more readable and maintainable but also allows for easier expansion and integration of new features in the future.

5. Practical Application of Python: Python's simplicity and readability make it an ideal choice for beginners. This project showcases how Python can be used effectively for developing functional applications, reinforcing its value as a versatile programming language.

11). Bibliography:-

* [Python Lists](<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>)
* [Python Dictionaries](<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>)
* [Real Python: Building a Simple To-Do List with Python](https://realpython.com/python-project-for-beginners/#building-a-simple-to-do-list)
* [GeeksforGeeks: To-Do List Application in Python](<https://www.geeksforgeeks.org/to-do-list-application-using-tkinter/>)
* "Automate the Boring Stuff with Python" by Al Sweigart: This book has a project that involves creating a simple command-line Todo list.
* "Python Crash Course" by Eric Matthes: This book covers basic Python projects, including a simple Todo list.
* [Simple To-Do List App](https://github.com/davidtavarez/simple-todo-list) - A basic implementation of a Todo list in Python.
* [Python Todo List](https://github.com/susan-sharma/Python-To-Do-List) - Another example of a Todo list application using Python and Tkinter.