TODO- LIST APPLICATION BASED ON STACK

MINOR PROJECT

- Institute Name: GH Patel College Of Engineering, Vallabh Vidyanagar, Anand
- Submitted to: MyJobGrow
- Submitted By: Meenakshi T.S

- Date Of Submission: 2/OCT/2025

Deployment link: https://github.com/nul-lhypothesis/TODO-List-Application-Using-Stack-Data-Structure

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **MyJobGrow** for guiding me throughout this internship and for providing me with the learning materials and project resources that helped me understand data structures and their real-world applications.
I also thank my mentors and the online tutors from MyJobGrow for their consistent support. Their guidance helped me strengthen my fundamentals in Python and apply them practically while developing this **TODO List Application**.
As a **first-year Computer Engineering student**, this project gave me a deeper understanding of how logic, structure, and coding come together to create a working system. I am grateful for this learning experience that has contributed to my growth as a budding programmer.

TABLE OF CONTENTS

INTRODUCTION TO PYTHON

Python is one of the most popular and powerful programming languages used in today's world. It is an interpreted, high-level, and general-purpose language that emphasizes readability and simplicity. Python was created by **Guido van Rossum** and first released in **1991**. The language was designed with the philosophy of making code easy to read and write, allowing programmers to express their ideas in fewer lines of code compared to other languages like C++ or Java.

Python supports multiple programming paradigms, including **procedural, object-oriented, and functional programming**, making it highly flexible for a wide range of applications. Its syntax is clean and straightforward, which helps beginners quickly understand programming concepts. This is one of the main reasons Python is widely taught in schools and colleges as a first programming language.

FEATURES OF PYTHON

Simple and easy to learn

Python’s syntax is clear and similar to english, making it very easy for beginners to learn and use.

Interpreted language

Python programs do not require compilation. The interpreter reads and executes the code line by line, making debugging simpler.

Object-oriented

Python supports object oriented programming concepts like classes, inheritance, polymorphism, which help in organizing and reusing code efficiently.

Extensive library support

Provides a large number of built-in libraries and modules that make it easy to perform various tasks such as file handling, data analysis, and web development.

Portable and open source

Python is available for free and can run on various platforms such as windows, macOS, linux without modification.

Dynamic Typing & large community support

In python we don’t need to declare the type of variable as type is automatically determined at run time and it also has a huge active global community that contributes to continuos development

APPLICATIONS OF PYTHON

Python is a versatile language used in almost every field of technology. Some of its major applications include:

- **Web Development:** Frameworks like Django and Flask are used to create powerful web applications.
- **Data Science and Machine Learning:** Libraries such as NumPy, Pandas, TensorFlow, and Scikit-learn make Python a leading language for data analysis and AI development.
- **Automation and Scripting:** Python is used for automating repetitive tasks such as file organization, email handling, and testing.
- **Software Development:** It is used to build desktop and mobile applications.
- **Game Development:** Libraries like Pygame help in creating simple 2D games.
- **Cybersecurity and Ethical Hacking:** Python scripts are used in security testing and penetration testing tools.

Python continues to grow in popularity due to its simplicity and wide range of applications. It bridges the gap between academic learning and real-world problem solving.

INTRODUCTION TO THE PROJECT

- The **TODO List Application** is a console-based Python program designed to help users organize and track their daily activities efficiently. It makes use of the **Stack data structure** to manage tasks — meaning the most recently added or completed task can be accessed or undone first, following the **Last In, First Out (LIFO)** principle.

- The project allows the user to add new tasks with details such as title, description, priority, and due date. Tasks can be marked as completed, deleted, or even undone using stack operations. It also provides sorting, searching, and viewing features that make it easy to organize tasks by priority, category, or date.

- All task data is saved in a **JSON file** so that the list remains available even after the program is closed. Through this project, I learned how to combine Python programming with **data structures** and **file handling** to build something both logical and practical.

CONCEPTS USED IN THE PROJECT

This project makes use of several important programming and data structure concepts that helped in building a functional and efficient TODO List Application. Each concept played a specific role — from organizing data and managing tasks to handling files and ensuring a smooth user experience. The key concepts applied in this project are explained below:

1. **Stack Data Structure:**
   Used for task management and undo operations. New tasks are added to the stack, and the most recently completed task can be undone using the LIFO approach.

2. **File Handling & JSON:**
   Enables saving and loading tasks from a file, ensuring the data is not lost when the program ends.

3. **Sorting Algorithm (Bubble Sort):**
   Implemented to organize tasks by different factors such as priority, due date, or category.

4. **Search Algorithm (Linear Search):**
   Used to find tasks quickly by keywords in title, description, or category.

5. **Object-Oriented Programming (OOP):**
   The entire application is built around the TODO List class, which makes the code more modular and readable.

6. **Data Structures (Lists & Dictionaries):**
   Tasks are stored as dictionaries inside lists for efficient data access and modification.

7. **User Interaction:**
   A simple text-based interface with menu options allows users to perform different operations smoothly.

8. **Error Handling:**
   The code uses validation and exception handling to avoid crashes and guide the user properly.

SOURCE CODE (screenshots)

todo list.py

```python
import datetime
import json
import os

DATA_FILE = "todo_data.json"

class TODOList:
    """
    Simple TODO List Application using Stack (student-coded).
    Main stacks:
      - self.tasks: list of task dicts (acts as main stack)
      - self.completed_stack: list of completed tasks (for undo)
    """

    def __init__(self, filename=DATA_FILE):
        self.filename = filename
        self.tasks = []
        self.completed_stack = []
        self.next_id = 1
        self.load_tasks()

    def load_tasks(self):
        try:
            if os.path.exists(self.filename):
                with open(self.filename, "r", encoding="utf-8") as f:
                    data = json.load(f)
                    self.tasks = data.get("tasks", [])
                    self.completed_stack = data.get("completed", [])
                    if self.tasks:
                        self.next_id = max(t["id"] for t in self.tasks) + 1
                print("Loaded existing tasks.")
```

```
32                else:
33                    print("No existing tasks found. Starting fresh.")
34            except Exception as e:
35                print(f"Error loading tasks (file may be corrupted): {e}")
36                self.tasks = []
37                self.completed_stack = []
38                self.next_id = 1
39
40        def save_tasks(self):
41            try:
42                with open(self.filename, "w", encoding="utf-8") as f:
43                    json.dump({"tasks": self.tasks, "completed": self.completed_stack}, f, indent=2)
44            except Exception as e:
45                print(f"Error saving tasks: {e}")
46
47        def reindex_ids(self):
48            """Reassign task IDs after deletion"""
49            for i, task in enumerate(self.tasks, start=1):
50                task["id"] = i
51            self.next_id = len(self.tasks) + 1
52
53        @staticmethod
54        def now_str():
55            return datetime.datetime.now().strftime("%d-%m-%Y %H:%M")
56
57        @staticmethod
58        def parse_date_str(date_str):
59            try:
60                return datetime.datetime.strptime(date_str, "%d-%m-%Y")
61            except Exception:
```

```
61            except Exception:
62                return datetime.datetime.max
63
64        @staticmethod
65        def truncate(text, n=40):
66            return text if len(text) <= n else text[:n-3] + "..."
67
68        def display_menu(self):
69            print("\n" + "="*50)
70            print("          TODO LIST APPLICATION ")
71            print("="*50)
72            print("1. Add New Task")
73            print("2. View All Tasks")
74            print("3. Mark Task as Completed")
75            print("4. Undo Last Completion")
76            print("5. Delete Task")
77            print("6. Sort Tasks")
78            print("7. Search Tasks")
79            print("8. View by Category")
80            print("9. View Statistics")
81            print("10. Exit")
82            print("-"*50)
83
84        def add_task(self):
85            print("\n--- Add New Task ---")
86            title = input("Enter task title: ").strip()
87            if not title:
88                print("Task title cannot be empty. Try again.")
89                return
90
```

```
90
91          description = input("Enter description (optional): ").strip()
92
93          print("\nPriority (1-High, 2-Medium, 3-Low) [default 2]: ", end="")
94          pr = input().strip()
95          priority = "High" if pr == "1" else "Low" if pr == "3" else "Medium"
96
97          print("\nCategory (work/personal/study/shopping/other) [default other]: ", end="")
98          cat = input().strip().title() or "Other"
99
100         due = input("Due date (DD-MM-YYYY) or leave blank: ").strip()
101         if due:
102             try:
103                 datetime.datetime.strptime(due, "%d-%m-%Y")
104             except Exception:
105                 print("Invalid due date format. It will be saved as blank.")
106                 due = ""
107
108         task = {
109             "id": self.next_id,
110             "title": title,
111             "description": description,
112             "priority": priority,
113             "category": cat,
114             "due_date": due,
115             "created_at": self.now_str(),
116             "completed": False
117         }
118         self.tasks.append(task)
119         self.next_id += 1
```

```
120             self.save_tasks()
121             print(f"Task '{title}' added (ID {task['id']}).")
122             input("Press Enter to continue...")
123
124         def view_tasks(self, tasks=None, header="All Tasks"):
125             if tasks is None:
126                 tasks = self.tasks
127             if not tasks:
128                 print("\nNo tasks to show.")
129                 return
130             print(f"\n--- {header} ---")
131             print("ID | Pri | Cat       | Status    | Title")
132             print("-"*60)
133             for t in tasks:
134                 status = "Done" if t.get("completed") else "Pending"
135                 title_display = self.truncate(t.get("title", ""), 38)
136                 print(f"{t['id']:2} | {t['priority'][0]}   | {t['category'][:9]:9} | {status:9} | {title_display}")
137             print("-"*60)
138
139         def complete_task(self):
140             if not self.tasks:
141                 print("\nNo tasks available to complete.")
142                 return
143             self.view_tasks()
144             raw = input("Enter Task ID to mark as completed: ").strip()
145             if not raw.isdigit():
146                 print("Enter a numeric ID.")
147                 return
148             tid = int(raw)
```

```
148             tid = int(raw)
149             for t in self.tasks:
150                 if t["id"] == tid:
151                     if t["completed"]:
152                         print("Task already completed.")
153                     else:
154                         self.completed_stack.append(t.copy())
155                         t["completed"] = True
156                         t["completed_at"] = self.now_str()
157                         self.save_tasks()
158                         print(f"Task '{t['title']}' marked as completed.")
159                     input("Press Enter to continue...")
160                     return
161             print("Task ID not found.")
162 
163         def undo_last_completion(self):
164             if not self.completed_stack:
165                 print("\nNo completed tasks to undo. Complete a task first.")
166                 return
167             last = self.completed_stack.pop()
168             for t in self.tasks:
169                 if t["id"] == last["id"]:
170                     t["completed"] = False
171                     t.pop("completed_at", None)
172                     self.save_tasks()
173                     print(f"Undid completion of task '{t['title']}'.")
174                     input("Press Enter to continue...")
175                     return
176             print("Unexpected: previously completed task not found.")
177 
```

```
176             print("Unexpected: previously completed task not found.")
177
178     def delete_task(self):
179         if not self.tasks:
180             print("\nNo tasks to delete.")
181             return
182         self.view_tasks()
183         try:
184             tid = int(input("Enter Task ID to delete: "))
185             for i, t in enumerate(self.tasks):
186                 if t["id"] == tid:
187                     deleted = self.tasks.pop(i)
188                     self.completed_stack = [c for c in self.completed_stack if c["title"] != deleted["title"]]
189                     self.reindex_ids()   # fixed: reindex IDs
190                     self.save_tasks()
191                     print(f"Deleted task '{deleted['title']}'.")
192                     return
193             print("Invalid Task ID.")
194         except ValueError:
195             print("Enter a valid number.")
196
197     def sort_tasks(self):
198         if not self.tasks:
199             print("\nNo tasks to sort.")
200             return
201         print("\nSort by:")
202         print("1. Priority (High -> Low)")
203         print("2. Due date (earliest first)")
204         print("3. Category (A-Z)")
205         print("4. Created date (newest first)")
206         choice = input("Choice (1-4): ").strip()
```

```
        choice = input("Choice (1-4): ").strip()
        arr = self.tasks.copy()

        n = len(arr)
        for i in range(n):
            for j in range(0, n - i - 1):
                swap = False
                if choice == "1":
                    order = {"High":3, "Medium":2, "Low":1}
                    if order[arr[j]["priority"]] < order[arr[j+1]["priority"]]:
                        swap = True
                elif choice == "2":
                    d1 = self.parse_date_str(arr[j].get("due_date",""))
                    d2 = self.parse_date_str(arr[j+1].get("due_date",""))
                    if d1 > d2:
                        swap = True
                elif choice == "3":
                    if arr[j]["category"].lower() > arr[j+1]["category"].lower():
                        swap = True
                elif choice == "4":
                    try:
                        c1 = datetime.datetime.strptime(arr[j]["created_at"], "%d-%m-%Y %H:%M")
                        c2 = datetime.datetime.strptime(arr[j+1]["created_at"], "%d-%m-%Y %H:%M")
                    except Exception:
                        c1 = c2 = datetime.datetime.min
                    if c1 < c2:
                        swap = True
                if swap:
                    arr[j], arr[j+1] = arr[j+1], arr[j]
        title = {
```

```
235            title = {
236                "1":"Tasks sorted by Priority",
237                "2":"Tasks sorted by Due date",
238                "3":"Tasks sorted by Category",
239                "4":"Tasks sorted by Created date"
240            }.get(choice, "Sorted Tasks")
241            self.view_tasks(arr, header=title)
242
243        def search_tasks(self):
244            if not self.tasks:
245                print("\nNo tasks to search.")
246                return
247            kw = input("Enter keyword to search (title/description/category): ").strip().lower()
248            if not kw:
249                print("No keyword entered.")
250                return
251            found = []
252            for t in self.tasks:
253                if (kw in t["title"].lower() or
254                    kw in t.get("description","").lower() or
255                    kw in t.get("category","").lower()):
256                    found.append(t)
257            self.view_tasks(found, header=f"Search results for '{kw}'")
258
259        def view_by_category(self):
260            if not self.tasks:
261                print("\nNo tasks to show.")
262                return
263            groups = {}
264            for t in self.tasks:
```

```
264             for t in self.tasks:
265                 cat = t.get("category","Other")
266                 groups.setdefault(cat, []).append(t)
267             for cat, items in groups.items():
268                 print(f"\n{cat} ({len(items)} tasks):")
269                 self.view_tasks(items, header=f"{cat} tasks")
270 
271     def view_statistics(self):
272         if not self.tasks:
273             print("\nNo tasks for statistics.")
274             return
275         total = len(self.tasks)
276         completed = sum(1 for t in self.tasks if t.get("completed"))
277         pending = total - completed
278         high = sum(1 for t in self.tasks if t.get("priority")=="High")
279         med = sum(1 for t in self.tasks if t.get("priority")=="Medium")
280         low = sum(1 for t in self.tasks if t.get("priority")=="Low")
281         rate = (completed / total * 100) if total else 0.0
282         print("\n--- Task Statistics ---")
283         print(f"Total tasks     : {total}")
284         print(f"Completed       : {completed}")
285         print(f"Pending         : {pending}")
286         print(f"High priority   : {high}")
287         print(f"Medium priority : {med}")
288         print(f"Low priority    : {low}")
289         print(f"Completion rate : {rate:.1f}%")
290 
291 def main():
292     app = TODOList()
293     while True:
```

```python
def main():
    app = TODOList()
    while True:
        app.display_menu()
        choice = input("Enter your choice (1-10): ").strip()
        if not choice:
            print("Please enter a valid option.")
            continue
        if choice == "1":
            app.add_task()
        elif choice == "2":
            app.view_tasks()
            input("Press Enter to continue...")
        elif choice == "3":
            app.complete_task()
        elif choice == "4":
            app.undo_last_completion()
        elif choice == "5":
            app.delete_task()
        elif choice == "6":
            app.sort_tasks()
            input("Press Enter to continue...")
        elif choice == "7":
            app.search_tasks()
            input("Press Enter to continue...")
        elif choice == "8":
            app.view_by_category()
            input("Press Enter to continue...")
        elif choice == "9":
            app.view_statistics()
            input("Press Enter to continue...")
```

```
            app.complete_task()
        elif choice == "4":
            app.undo_last_completion()
        elif choice == "5":
            app.delete_task()
        elif choice == "6":
            app.sort_tasks()
            input("Press Enter to continue...")
        elif choice == "7":
            app.search_tasks()
            input("Press Enter to continue...")
        elif choice == "8":
            app.view_by_category()
            input("Press Enter to continue...")
        elif choice == "9":
            app.view_statistics()
            input("Press Enter to continue...")
        elif choice == "10":
            print("\nSaving and exiting. Goodbye!")
            app.save_tasks()
            break
        else:
            print("Invalid choice. Please enter a number from 1 to 10.")

if __name__ == "__main__":
    main()
```

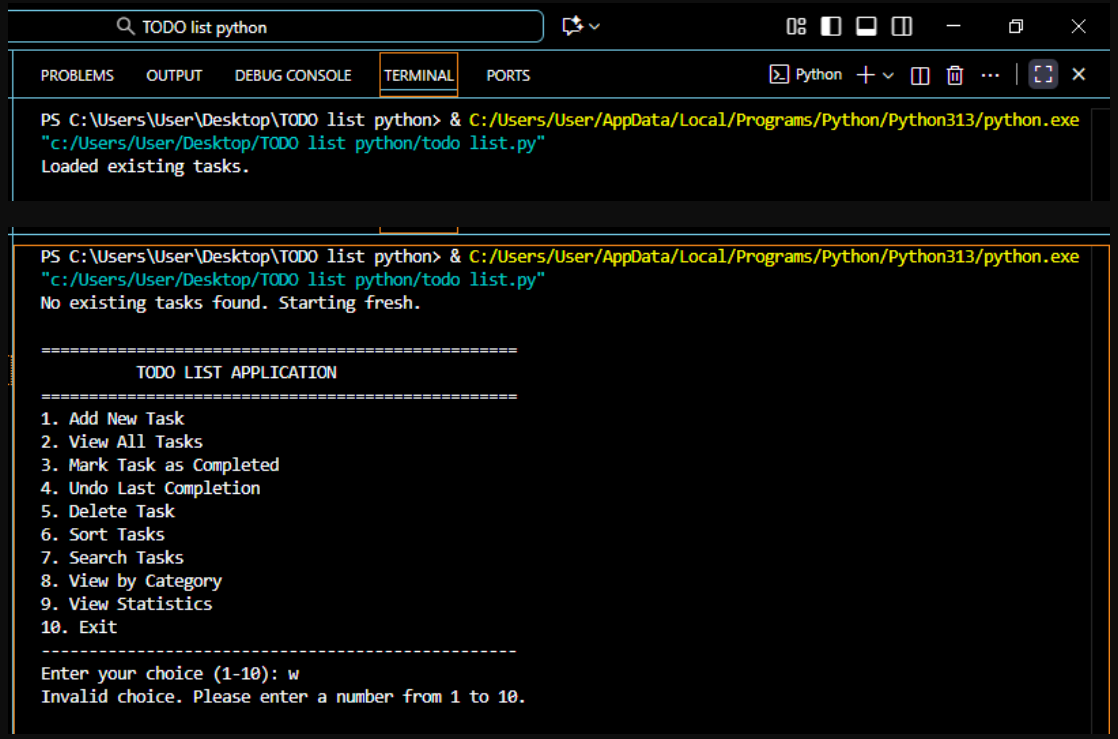
WORKFLOW OF THE CODE

The workflow of the TODO-List follows a clear, step-by-step process:

1. **Start & Load**
   When the program starts, it checks if the todo_data.json file exists. If it does, the saved tasks are loaded into memory. If not, the program starts with an empty task list.
2. **Show Main Menu**
   The main menu is displayed with options like Add, Display, Complete, Undo, Delete, Sort, Search, View by Category, Statistics, and Exit.
3. **Add Task (Push)**
   User enters task details (title, description, priority, category, due date). The task is stored as a dictionary and **pushed** onto the main stack. The JSON file is updated immediately.
4. **Display Tasks**
   The program prints all current tasks in a neat format showing ID, priority, category, status, and creation date.
5. **Complete Task (Copy → Completed Stack)**
   When a task is marked completed, a copy is saved to the completed_stack and the task status is updated. This enables undoing later.

6. **Undo Last Completion (Pop)**
   The last completed task is **popped** from the completed_stack and its status is set back to pending in the main stack — LIFO behavior in action.
7. **Delete Task**
   User can remove any task by ID. The task is deleted from the main stack and the JSON file is updated.
8. **Sort Tasks (Bubble Sort)**
   User can sort tasks by priority, due date, category, or creation date. Sorting is done with a bubble-sort implementation and the sorted list is displayed.
9. **Search Tasks (Linear Search)**
   The program searches tasks by keyword (title, description, or category) using linear search and displays matching results.
10. **View by Category (Group View)**
    Tasks are grouped and shown by category (Work / Personal / Study / etc.) in a simple tree-like list, with counts for each category.
11. **View Statistics**
    Shows total tasks, completed vs pending counts, priority breakdown (High/Medium/Low), and a completion percentage.
12. **Save & Exit**
    On exit, all changes are ensured to be saved in the JSON file and the program displays a thank-you message before closing.

OUTPUT EXPLANATION



1. Program start & Main Menu:

•When the program is first run, it checks if a saved data file (todo_data.json) already exists from a previous session.

•If found, it shows “Loaded existing tasks.”, else displays “No existing tasks found. Starting fresh.”

•Then the main menu of the TODO List appears with all options from 1 to 10.

•The user can choose any option, and invalid input displays “Invalid choice. Please enter a number from 1 to 10.”

```
--- Add New Task ---
Enter task title: do laundry
Enter description (optional):

Priority (1-High, 2-Medium, 3-Low) [default 2]: 2

Category (work/personal/study/shopping/other) [default other]: personel
Due date (DD-MM-YYYY) or leave blank: 4/11/2025
Invalid due date format. It will be saved as blank.
Task 'do laundry' added (ID 2).
Press Enter to continue...
```


```
--- Add New Task ---
Enter task title: do laundry
Enter description (optional):

Priority (1-High, 2-Medium, 3-Low) [default 2]: 2

Category (work/personal/study/shopping/other) [default other]: personel
Due date (DD-MM-YYYY) or leave blank: 4/11/2025
Invalid due date format. It will be saved as blank.
Task 'do laundry' added (ID 2).
Press Enter to continue...
```


2. Adding a task :

- This output shows the process of adding a new task into the list.
- The user enters details such as task title, description, priority level, category, and due date.
- The program adds the entered task to the stack and confirms successful addition.
- If the due date format isn't correct, it shows "Invalid due date format. It will be saved as blank" and keeps that field empty in the JSON file.

```
================================================
         TODO LIST APPLICATION
================================================
1. Add New Task
2. View All Tasks
3. Mark Task as Completed
4. Undo Last Completion
5. Delete Task
6. Sort Tasks
7. Search Tasks
8. View by Category
9. View Statistics
10. Exit
------------------------------------------------
Enter your choice (1-10): 2

--- All Tasks ---
ID | Pri | Cat       | Status    | Title
------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | H   | Study     | Pending   | submit record
------------------------------------------------------------
Press Enter to continue...
```

3. Display(view) all tasks :

- When the user selects option 2, all current tasks are displayed neatly in a tabular format.
- The table includes columns like ID, Priority, Category, Status, and Title.
- Tasks are listed in the order they were added, showing the LIFO nature of the stack.
- If no tasks are found, the program shows “No tasks to show” to indicate an empty list.

```
------------------------------------------------
Enter your choice (1-10): 3

--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | H   | Study     | Pending   | submit record
-------------------------------------------------------------
Enter Task ID to mark as completed: 2
Task 'do laundry' marked as completed.
Press Enter to continue...
```

```
------------------------------------------------
Enter your choice (1-10): 2

--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Done      | do laundry
 3 | H   | Study     | Pending   | submit record
-------------------------------------------------------------
Press Enter to continue...
```

```
Enter Task ID to mark as completed: 2
Task already completed.
Press Enter to continue...
```

4. Marking task as completed :

- This feature is used to mark a specific task as completed using its ID.
- Once confirmed, the task status changes from “Pending” to “Done” and gets time-stamped.
- The completed task is also stored in a separate stack, allowing it to be undone later.
- If a user selects an already completed task, the system responds with “Task already completed.”

```
--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Done      | do laundry
 3 | H   | Study     | Pending   | submit record
-------------------------------------------------------------
```

```
------------------------------------------------
Enter your choice (1-10): 4
Undid completion of task 'do laundry'.
Press Enter to continue...
```

```
--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | H   | Study     | Pending   | submit record
-------------------------------------------------------------
Press Enter to continue...
```

```
------------------------------------------------
Enter your choice (1-10): 4

No completed tasks to undo. Complete a task first.
```

5. Undoing the last completed task:

- This output shows how the undo feature works using the LIFO property of a stack.
- The last completed task is restored to “Pending,” reversing its completion.
- It’s useful if a task was marked done by mistake or too early.
- If there are no completed tasks, the message “No completed tasks to undo. Complete a task first.” is displayed.

```
------------------------------------------------
Enter your choice (1-10): 5

--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | H   | Study     | Pending   | submit record
-------------------------------------------------------------
Enter Task ID to delete: 3
Deleted task 'submit record'.
```

```
------------------------------------------------
Enter your choice (1-10): 2

--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
-------------------------------------------------------------
Press Enter to continue...
```

```
------------------------------------------------
Enter your choice (1-10): 5

No tasks to delete.
```

6. Deleting a task :

- This output demonstrates how tasks can be permanently deleted from the list.
- The user enters the ID of the task they wish to delete.
- Once removed, the remaining tasks are automatically reindexed for proper numbering.
- If there are no tasks available to delete, it displays “No tasks to delete” to notify the user.

```
--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | M   | Other     | Pending   | buy prints
 4 | M   | Other     | Pending   | practice piano
-------------------------------------------------------
```

```
------------------------------------------------
Enter your choice (1-10): 6

Sort by:
1. Priority (High -> Low)
2. Due date (earliest first)
3. Category (A-Z)
4. Created date (newest first)
Choice (1-4): 3

--- Tasks sorted by Category ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------
 3 | M   | Other     | Pending   | buy prints
 4 | M   | Other     | Pending   | practice piano
 2 | M   | Personel  | Pending   | do laundry
 1 | H   | Study     | Pending   | finish lab work
-------------------------------------------------------
Press Enter to continue...
```

7. Sorting tasks :

- The sorting option allows tasks to be arranged by Priority, Due Date, Category, or Created Date.
- Sorting is implemented using the Bubble Sort algorithm to maintain a student-coded feel.
- Before sorting, tasks appear in the order they were entered.
- After sorting, tasks rearrange based on the user's chosen criteria, like High to Low priority or by category alphabetically.

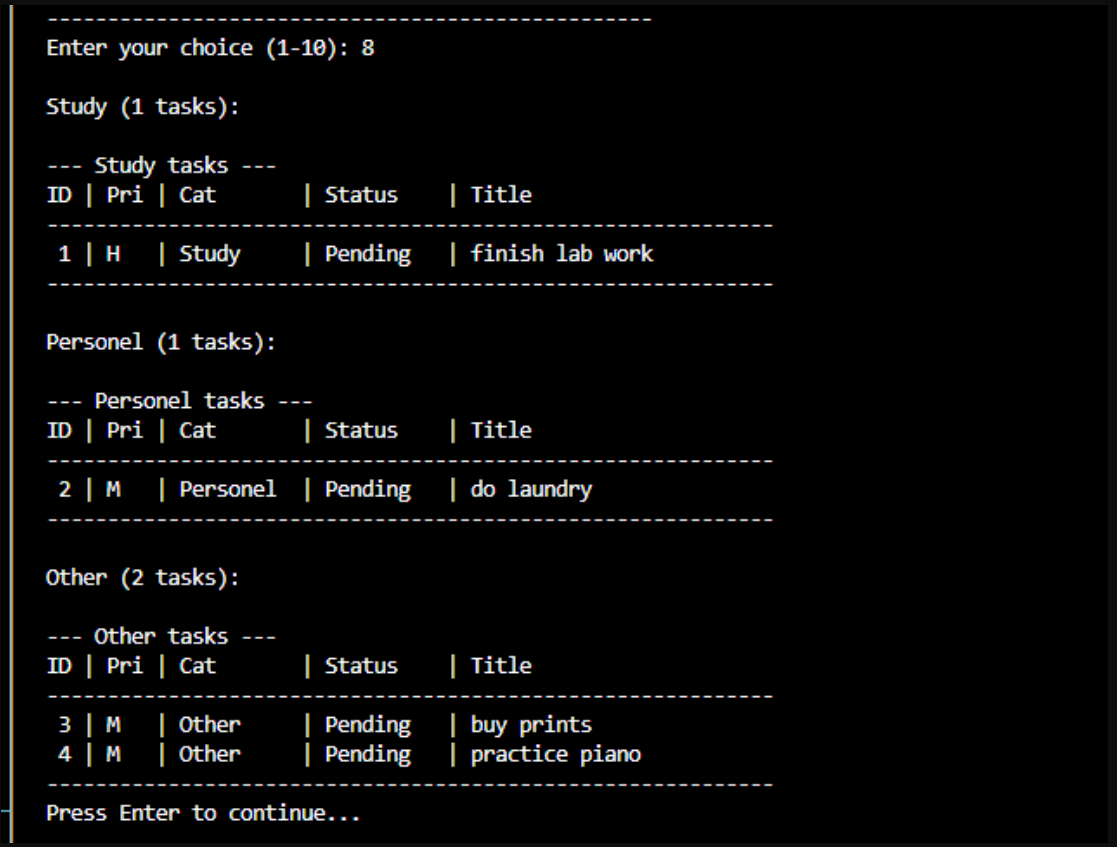
```
================================================
         TODO LIST APPLICATION
================================================
1. Add New Task
2. View All Tasks
3. Mark Task as Completed
4. Undo Last Completion
5. Delete Task
6. Sort Tasks
7. Search Tasks
8. View by Category
9. View Statistics
10. Exit
------------------------------------------------
Enter your choice (1-10): 7
Enter keyword to search (title/description/category): lab

--- Search results for 'lab' ---
ID | Pri | Cat       | Status    | Title
------------------------------------------------------------
 1 | H   | Study     | Pending   | finish lab work
------------------------------------------------------------
Press Enter to continue...
```

```
No tasks to show.
Press Enter to continue...
```

8. Searching for a task:

- This shows the search feature where the user enters a keyword to find specific tasks.
- The program performs a linear search through the titles, descriptions, and categories.
- Only the tasks containing the keyword are displayed as search results.
- If no matches are found or the list is empty, it shows “No tasks to search.”



```
------------------------------------------------
Enter your choice (1-10): 8

Study (1 tasks):

--- Study tasks ---
ID | Pri | Cat      | Status    | Title
-------------------------------------------------------------
 1 | H   | Study    | Pending   | finish lab work
-------------------------------------------------------------

Personel (1 tasks):

--- Personel tasks ---
ID | Pri | Cat      | Status    | Title
-------------------------------------------------------------
 2 | M   | Personel | Pending   | do laundry
-------------------------------------------------------------

Other (2 tasks):

--- Other tasks ---
ID | Pri | Cat      | Status    | Title
-------------------------------------------------------------
 3 | M   | Other    | Pending   | buy prints
 4 | M   | Other    | Pending   | practice piano
-------------------------------------------------------------
Press Enter to continue...
```

9. Viewing tasks by category:

- This output displays how tasks are grouped and shown based on their categories.
- For example, Work, Study, or Personal tasks appear under their own sections.
- Each category displays its total number of tasks and their details.
- This helps in organizing tasks efficiently and checking progress category-wise.

```
--- All Tasks ---
ID | Pri | Cat       | Status    | Title
-------------------------------------------------------
 1 | H   | Study     | Done      | finish lab work
 2 | M   | Personel  | Pending   | do laundry
 3 | M   | Other     | Pending   | buy prints
 4 | M   | Other     | Pending   | practice piano
-------------------------------------------------------
```

```
-------------------------------------------------
Enter your choice (1-10): 9

--- Task Statistics ---
Total tasks     : 4
Completed       : 1
Pending         : 3
High priority   : 1
Medium priority : 3
Low priority    : 0
Completion rate : 25.0%
Press Enter to continue...
```

10. Viewing statistics :

- This screen displays overall task statistics in a summarized format.
- It shows total tasks, completed and pending counts, and the number of tasks per priority level.
- The completion percentage is calculated and displayed clearly.
- This summary helps track progress and productivity at a glance.

```
----------------------------------------------------
Enter your choice (1-10): 10

Saving and exiting. Goodbye!
PS C:\Users\User\Desktop\TODO list python>
```

{} todo_data.json > ...


```json
{
  "tasks": [
    {
      "id": 1,
      "title": "finish lab work",
      "description": "write conclusion",
      "priority": "High",
      "category": "Study",
      "due_date": "5-11-2025",
      "created_at": "01-11-2025 15:25",
      "completed": true,
      "completed_at": "01-11-2025 15:42"
    },
    {
      "id": 2,
      "title": "do laundry",
      "description": "",
      "priority": "Medium",
      "category": "Personel",
      "due_date": "",
      "created_at": "01-11-2025 15:26",
      "completed": false
```


11. Save and exit :

- When the user selects the Exit option, the program saves all the tasks automatically.
- All task data is stored inside the file `todo_data.json` for future sessions.
- This ensures that even after closing, the data remains safe and reusable.
- The file structure can be opened anytime to view saved tasks in JSON format.

CONCLUSION

Through this project, I got a better understanding of how Python and data structures can actually be used together to build something useful. While making this To-Do List program, I applied what I learned about lists, stacks, sorting, and file handling in a more practical way. I also got to see how storing data in JSON files helps in keeping everything saved even after the program is closed.

Overall, this project was a good learning experience for me. I made mistakes, fixed them, and learned how to structure code properly. It also made me realize how concepts we study can be connected and used in real applications.

BIBLIOGRAPHY

•**Python Official Documentation** – https://docs.python.org/3/

• **W3Schools Python Tutorials** – https://www.w3schools.com/python/

• **GeeksforGeeks (Data Structures & Algorithms)** – https://www.geeksforgeeks.org/

• **Stack Overflow** – for discussions, debugging help, and community-shared logic examples

• **Internship Learning Materials and sessions, college notes and reference materials.**