

This was a self-learning assignment. The students created their own assignment, following certain guidelines. Once approved, we completed this assignment. The assignment needed to cover a range of topics. This will showcase our programming skills and proficiency.

# Self-Study Learning Outcomes

Python Language

Peter Vaughan

---

## Contents

Name Manager System Project.....	2
Skills Covered .....	2
Project Overview .....	2
Pickle Code Example .....	3
Shelve Code Example .....	3
How to Run the Program .....	3
Submission Expectation .....	4
Output.....	5
Submission .....	9

## Name Manager System Project

**Objective:** Build a modular Python application that allows users to manage (add, view, search, and delete) names in a persistent .dat file using object-oriented programming principles. Photos are to be included in file submission.

---

### Skills Covered

<u>Skill</u>	<u>Description</u>
<b>Basic Syntax</b>	Variables, loops, conditionals
<b>Functions</b>	Modular code organization
<b>File I/O</b>	Read/write binary files. Bonus if using pickle or shelve
<b>OOP</b>	Classes, inheritance, encapsulation, polymorphism
<b>Exception Handling</b>	Robust error catching
<b>Multiple Files</b>	Modular structure using packages or simple modules
<b>CLI Interaction</b>	Input/output in the terminal
<b>Data Validation</b>	Ensuring valid names only

---

### Project Overview

Create a command-line tool for managing a database of names. The program supports:

1. Adding a name
  2. Viewing all names
  3. Searching for a name
  4. Deleting a name
  5. Saving and loading names from a .dat file
-

## Pickle Code Example

```
import pickle

# Suppose you have a list of Person objects
people = [Person("Alice", "Smith"), Person("Bob", "Jones")]

# Saving to a file (serialization)
with open('names.dat', 'wb') as file:
    pickle.dump(people, file)

# Loading from a file (deserialization)
with open('names.dat', 'rb') as file:
    loaded_people = pickle.load(file)
```

---

## Shelve Code Example

```
import shelve

# Suppose you have a list of Person objects
people = [Person("Alice", "Smith"), Person("Bob", "Jones")]

# Saving to shelve file
with shelve.open('names_shelve') as db:
    db['people'] = people # store the list under the key 'people'

# Loading from shelve file
with shelve.open('names_shelve') as db:
    loaded_people = db['people']
```

---

## How to Run the Program

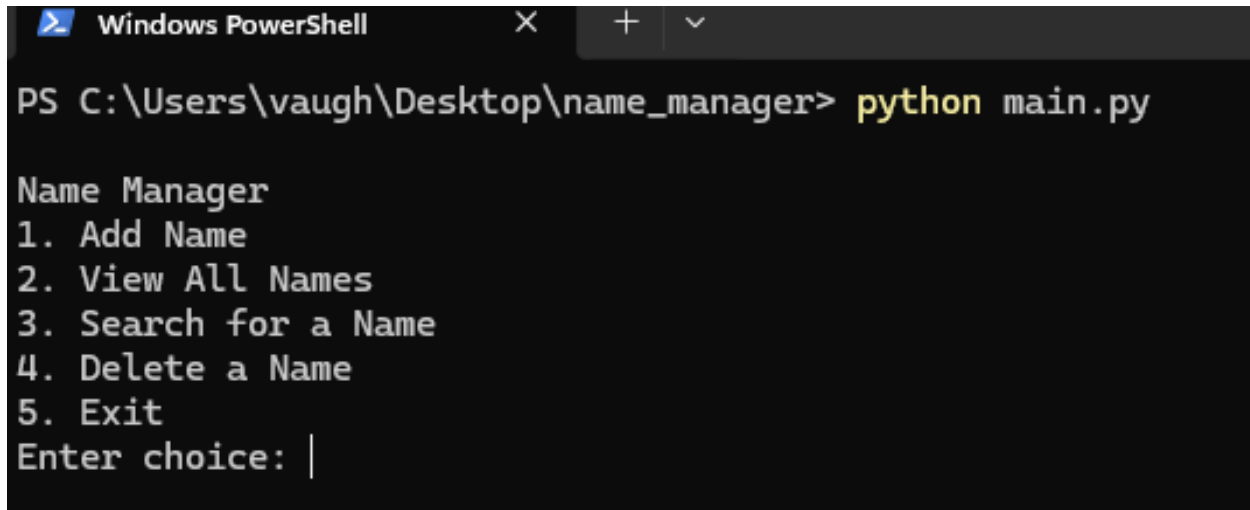
1. Save all files in the same name\_manager folder
2. Open terminal or command prompt
3. Navigate to the folder
4. Run: python main.py

## Submission Expectation

- GitHub Link with the following:
  - Code
  - README.md
- README.md (on GitHub):
  - Project title and description
  - Setup instructions
  - How to run/demo the project
- Photos
  - Screenshots of your application in action
  - Optional: photos of setup/hardware (if applicable)
- Documentation (typically a PDF or DOCX):
  - Project overview and goals
  - System architecture / design diagrams
  - Implementation details (including technologies used)
  - The IDE and how to run the code
  - Testing strategy and results – including testing units
  - Any challenges faced and how they were solved
  - Future work or improvements

## Output

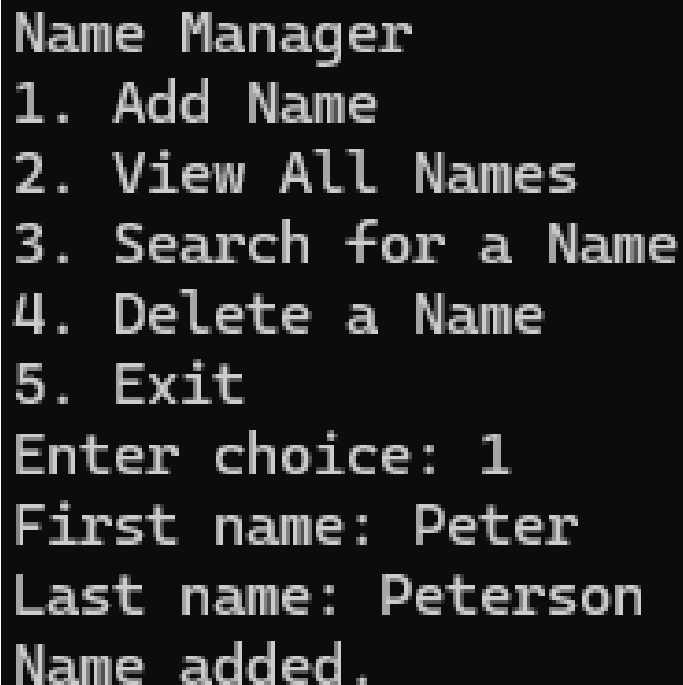
I started the program through the command lines as the IDE. This is one of many ways to use the code. However, as per assignment, I started the code through the command lines.



```
Windows PowerShell
PS C:\Users\vaugh\Desktop\name_manager> python main.py

Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: |
```

I can add a first name and last name, which is stored in a .dat file. This program could be expanded to do error-handling if we had parameters. This could prove challenging for me depending on the parameters. While we did learn some error-handling, some advanced concepts might be difficult at first.



```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 1
First name: Peter
Last name: Peterson
Name added.
```

We can view the full list of names. If there is only one name, it will only show one name. In a future example, if there are no names then the program will not display anything. We could format the name differently. In my scenario, it is first name and last name. We could do last name then first name.

```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 2
- Peter Peterson
```

We can search for a name in the .dat file. Results are listed. We would find and list all names that are requested. For example, if there was two different Peter's, then both would be listed.

```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 3
Search query: tyu
No matches found.
```

```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 3
Search query: Peter
Matches found:
- Peter Peterson
```

We can delete a name or multiple names. For this program, we are deleting everyone with the given name. If we were to improve or modify the program and to avoid confusion if people have the same first name is that the program could have been approached in different way. One example is to have an ID associated to the name. Then we can delete by ID. By default, there is no ID associated to the names. Another example is to delete by name position. When we check the name by viewing all the names, we could delete the name by position. For example, if we wanted to delete the third name, we could enter "3" to delete the third name regardless of the name. then we can verify if we wanted to delete a name an provide the given name.

```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 4
Name to delete: Peter
1 name(s) deleted.
```









```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 2
```



Our last option in our program is exiting. This exits the program with a message. I kept to a simple, yet effective message.

```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 5
Goodbye!
PS C:\Users\vaugh\Desktop\name_manager> |
```

This showcases my multiple python files along with the Dat file and photos folder. The ReadMe File will be created in the GitHub Repo. The “\_pycache\_” and “names” data file are created after you run the program after the first time. This will be re-created after each time you run the program. We could create a different file each time, following a certain naming convention.

 _pycache_	2025-05-02 1:11 PM	File folder	
 Photos	2025-05-05 8:04 PM	File folder	
 file_handler	2025-05-02 1:06 PM	Python Source File	1 KB
 main	2025-05-02 1:08 PM	Python Source File	2 KB
 manager	2025-05-02 1:07 PM	Python Source File	1 KB
 Name Manager System Project	2025-05-05 2:35 PM	Microsoft Word D...	20 KB
 name_model	2025-05-02 1:05 PM	Python Source File	1 KB
 names	2025-05-05 8:02 PM	DAT File	1 KB

# Submission

GitHub Link: <https://github.com/Vaughan-Peter/Python-SelfLearning>

ReadMe Documentation: Done

Python-SelfLearning /

Vaughan-Peter Create README.md

Name	Last commit message	Last commit date
Photos	My Learning	4 minutes ago
Name Manager System Project.docx	My Learning	4 minutes ago
README.md	Create README.md	now
file_handler.py	My Learning	4 minutes ago
main.py	My Learning	4 minutes ago
manager.py	My Learning	4 minutes ago
name_model.py	My Learning	4 minutes ago

README.md

How to Run the Program

1. Save all files in the same name\_manager folder.
2. Open terminal or command prompt.
3. Navigate to the folder.
4. Run: python main.py

For example:

```
Windows PowerShell
PS C:\Users\vaugh\Desktop\name_manager> python main.py

Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: |
```

Test Units / Unit Testing: Not used as program was basic. If this program got more complicated, unit testing could be used to test certain aspects.