Sabrina Fechtner

5 December 2023

IT FDN 110 B Au 23

Assignment 08

Employee Ratings

# Description

This program allows the user to input and record employee ratings. This program allows the user to input first and last name, a date of review, and a review rating for multiple inputs and writes information into a JSON file. In addition, this program is amendable and can be relaunched at a later time to add additional employee ratings. In addition, validation/test files are also included in this program to make sure the program integrity over time.

# Table of Contents

The

## Installation

There are three .py files for the program:

1. main.py
   1. main body to run the program
2. presentations\_classes.py
   1. contains input/output processing
      1. error messages
      2. menu
      3. menu choice
      4. input employee data
      5. output employee data
3. processing\_classes.py
   1. contains Person(Employee) classes and file processing (read/write to file)

Import .py files using any IDE that can open .py files. However, for ease of access PyCharm is preferred since this program was written using it.

# **Usage**

Provide instructions and examples so users/contributors can use the project. This will make it easy for them in case they encounter a problem – they will always have a place to reference what is expected.

You can also make use of visual aids by including materials like screenshots to show examples of the running project and also the structure and design principles used in your project.

Also if your project will require authentication like passwords or usernames, this is a good section to include the credentials.

# **Credits**

Thank you to Randal Root for providing examples to work off of.

This program was made for IT FDN 110 B Au 23: Foundations Of Programming: Python course. Thank you, instructors, for their time and effort for the course.

Author: Sabrina Fechtner github: <https://github.com/sfechtner42?tab=repositories>

# **Examples**

The file function class still has the same operations from last week where the program would: 1) open/read and append (Figure 5) or 2) create and write to (Figure 6). This week, these static methods tracked a list of student objects instead of dictionaries defined as student. In doing so, these functions were modified so that the json will still load a list of dictionaries within the student object using manual integration.

# **Contributing**

Next, the IO functions were pulled and modified from last week. The general error handling operator and the menu IO operators (Figure 7) were unchanged from last week. The largest changes were in the student data IO operators so that they use the student object. The output student data method itterates over each varible in the student class by assigning it to another str variable (Figure 8). The input student data method uses a similar method to Figure 5 where the student object is maniupated into a list of dictionaries that can be saved onto the opened/created json file. Because this data is part of the student class, input is also subject to validataion in Figures 3 and 4 before being saved (Figure 9).

# License

Now the main function calls all the defined functions (Figure 10). The only major change from last week is that now I defined the students variable as a list of the Student object.

## Credits

I ran my program successfully in PyCharm and in the terminal (Figure 11) where I registered myself for Python 100 and my cat, Cinnamon, for Python 200. I intentionally entered symbols to confirm that the error handling worked. I also entered all the data in lowercase and was happy to see the display and save file had all the inputs correctly capitalized. Both methods of running the code produced a .JSON file showing Cinnamon and I registered for our respective Python courses (Figure 12).

# Summary

This week, I modified my interactive course registration program from last week where I incorporated two new object classes with their own validation code. As highlighted in the lecture, separating things by concern generally results in the main program being unchanged. I came to appreciate how separating things into classes helps greatly with debugging. If I made a mistake or an error came up with a certain operation, it was much easier to look at only a section/operation rather than my entire code.

Description:

Offer a brief description of your project, outlining its purpose, functionality, and key features.

Table of Contents:

Include a table of contents with links to different sections in your README for easy navigation.

Installation:

Provide instructions on how to install and set up your project. Include any dependencies that need to be installed and steps to configure the environment.

Usage:

Explain how to use your project. Provide examples, command-line instructions, or code snippets to guide users on using different features.

Configuration:

If your project involves configuration settings, provide information on how users can configure the system to suit their needs.

Examples:

Include examples or use cases to demonstrate how your project can be applied. This could involve sample input/output or screenshots.

Contributing:

If you want to encourage contributions to your project, provide guidelines on how others can contribute. Include information on submitting issues, pull requests, and any coding standards you follow.

License:

Specify the license under which your project is distributed. This is essential for communicating how others can use, modify, and distribute your code.

Credits:

Acknowledge contributors, libraries, or resources that played a role in your project.

Contact Information:

Provide your contact information or ways for users to reach out with questions, feedback, or issues.

Version History:

Document changes, updates, and releases with a version history. Include dates, version numbers, and a brief summary of changes.

Additional Documentation:

If your project has additional documentation, provide links or references to other documents that users may find helpful.

Known Issues:

List any known issues or limitations of your project. This helps manage user expectations and provides transparency.

FAQ (Frequently Asked Questions):

Include a section for frequently asked questions to address common queries.

Dependencies:

List the external libraries, frameworks, or tools your project relies on.

Security:

If applicable, provide information on any security considerations or best practices for securing your project.

Testing:

If your project includes testing procedures, explain how users can run tests and contribute to testing efforts.