Name: Samuel Foley
Date: March 13, 2024
Course: IT FDN 110A
Github: https://github.com/Samuel-a-m-f/Intro_to_Prog_Python_Mod08

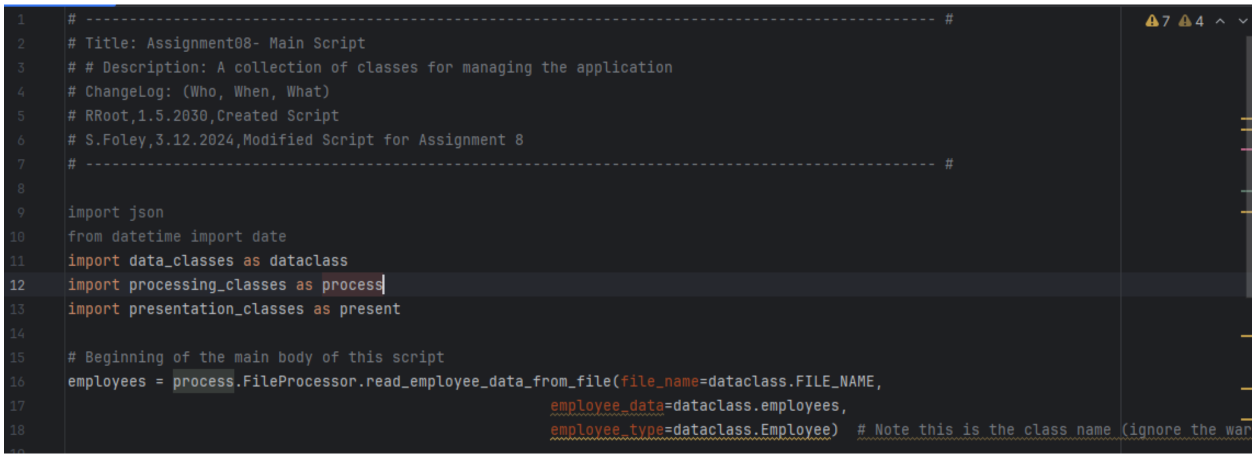
Assignment 08 – Creating Applications

Introduction

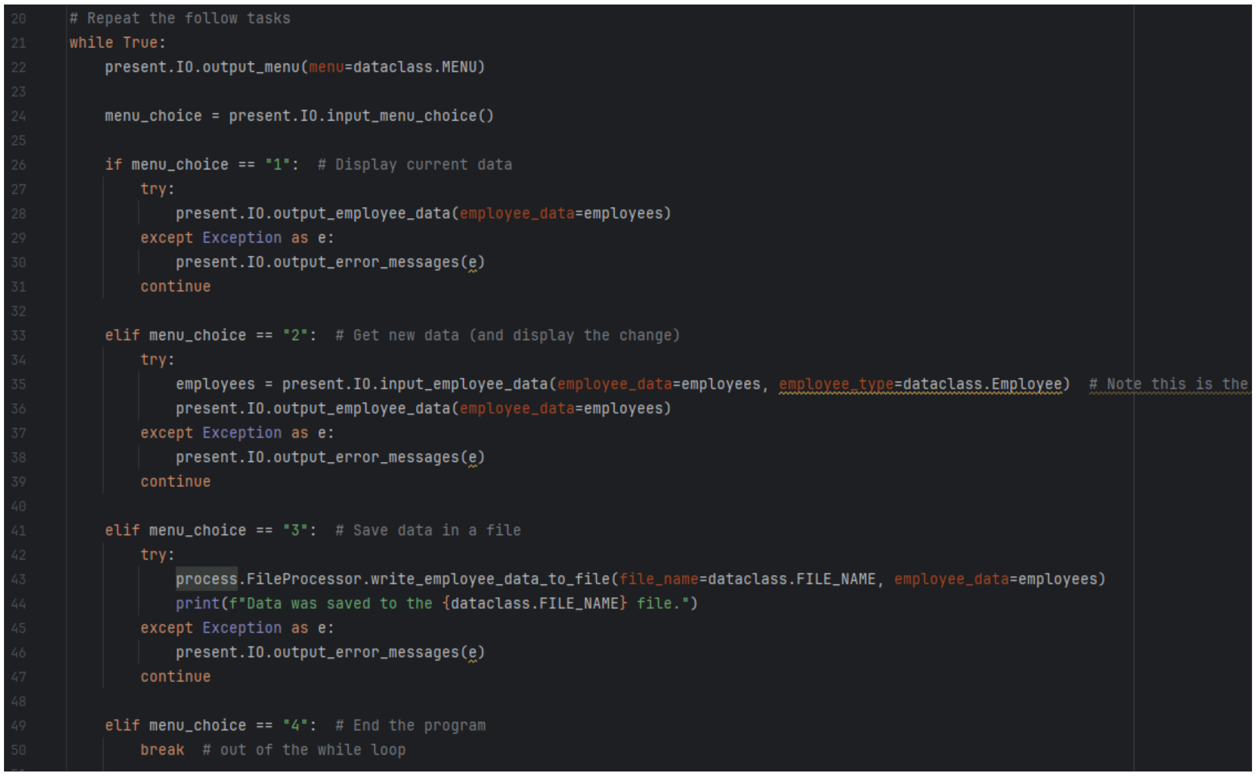
This document is going to describe the steps that I took for this week's assignment. This assignment is the culmination of all of the lessons before and creates an application using classes and modules. The overall goal is to create an employee application to register an employee, their review date and their review score. The application will be broken into a different file for main, presentation_classes, data_classes, and processing_classes. There are also going to be test classes for test_data_classes, test_presentation_classes, and test_processing_classes.

Main.py

The main.py portion of this application is fairly simple and now houses just the while loop to call IO under presentation_classes, and processing_classes. The presentation_classes and processing_classes are both imported at the very beginning of the scrip and will be referred to as present and process, respectively.



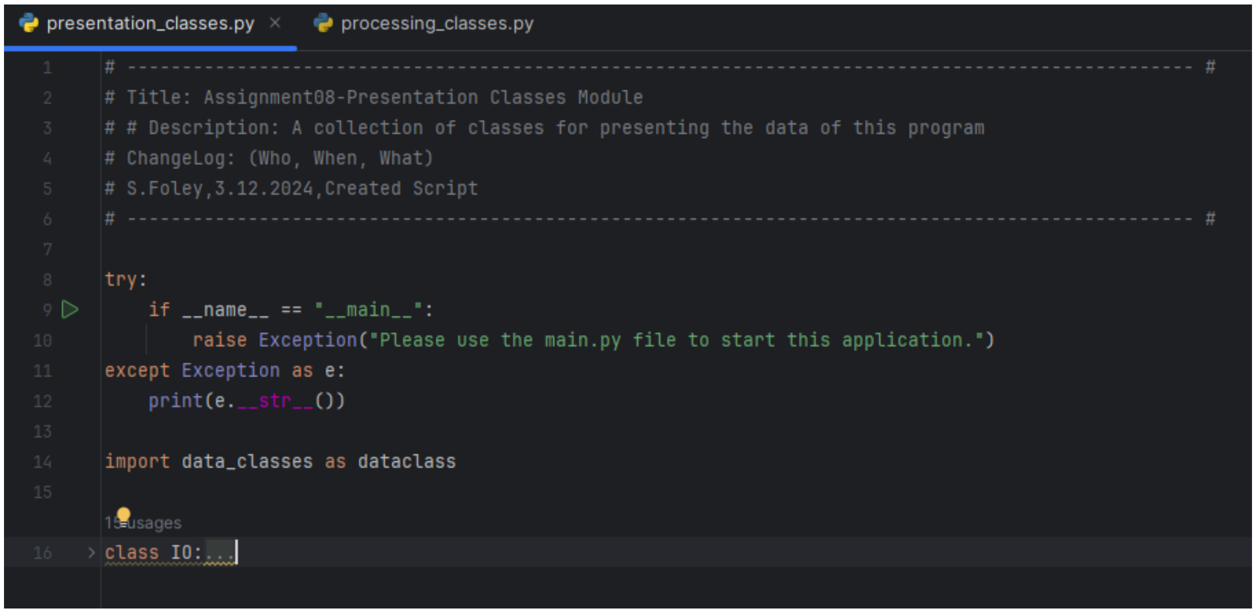
```
# ---------------------------------------------------------------------------------------------------- #
# Title: Assignment08- Main Script
# # Description: A collection of classes for managing the application
# ChangeLog: (Who, When, What)
# RRoot,1.5.2030,Created Script
# S.Foley,3.12.2024,Modified Script for Assignment 8
# ---------------------------------------------------------------------------------------------------- #

import json
from datetime import date
import data_classes as dataclass
import processing_classes as process
import presentation_classes as present

# Beginning of the main body of this script
employees = process.FileProcessor.read_employee_data_from_file(file_name=dataclass.FILE_NAME,
                                                               employee_data=dataclass.employees,
                                                               employee_type=dataclass.Employee)  # Note this is the class name (ignore the war
```



```
# Repeat the follow tasks
while True:
    present.IO.output_menu(menu=dataclass.MENU)

    menu_choice = present.IO.input_menu_choice()

    if menu_choice == "1":  # Display current data
        try:
            present.IO.output_employee_data(employee_data=employees)
        except Exception as e:
            present.IO.output_error_messages(e)
        continue

    elif menu_choice == "2":  # Get new data (and display the change)
        try:
            employees = present.IO.input_employee_data(employee_data=employees, employee_type=dataclass.Employee)  # Note this is the
            present.IO.output_employee_data(employee_data=employees)
        except Exception as e:
            present.IO.output_error_messages(e)
        continue

    elif menu_choice == "3":  # Save data in a file
        try:
            process.FileProcessor.write_employee_data_to_file(file_name=dataclass.FILE_NAME, employee_data=employees)
            print(f"Data was saved to the {dataclass.FILE_NAME} file.")
        except Exception as e:
            present.IO.output_error_messages(e)
        continue

    elif menu_choice == "4":  # End the program
        break  # out of the while loop
```

Presentation_classes.py

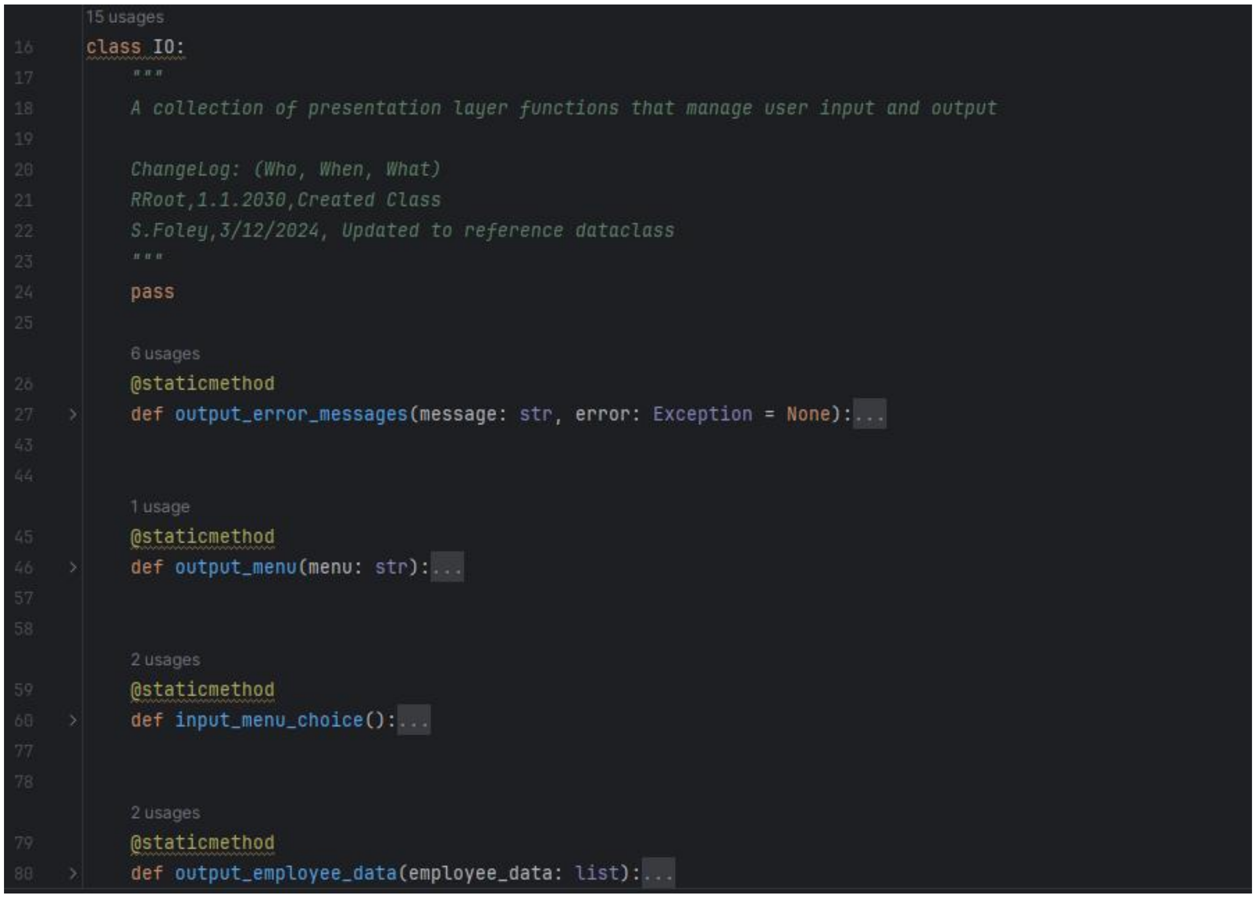
This module covers the IO class. The only difference between the previous IO version and this current one is that there is an import of the data_classes module at the top, and if this module is attempted to be run, there is an error telling you to run the main.py instead.



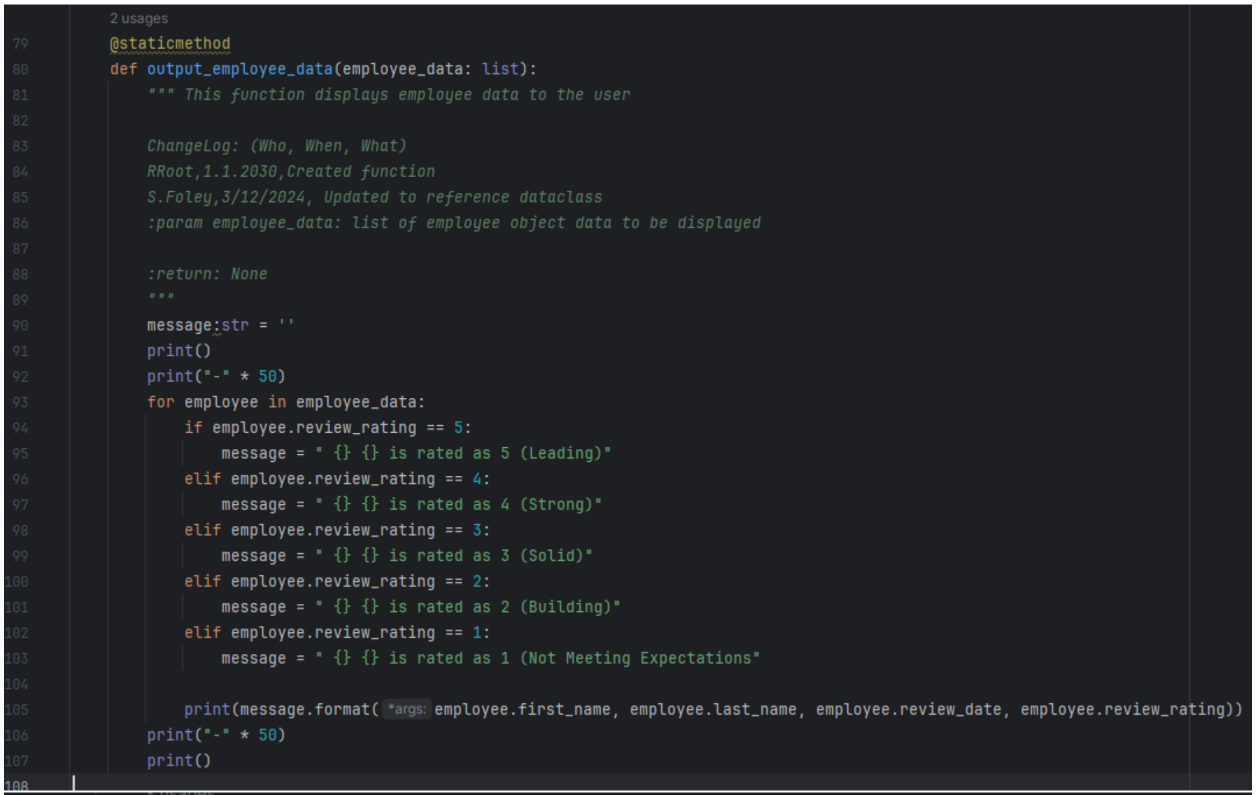
presentation_classes.py × | processing_classes.py

```
# ---------------------------------------------------------------------------------------------------- #
# Title: Assignment08-Presentation Classes Module
# # Description: A collection of classes for presenting the data of this program
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# ---------------------------------------------------------------------------------------------------- #

try:
    if __name__ == "__main__":
        raise Exception("Please use the main.py file to start this application.")
except Exception as e:
    print(e.__str__())

import data_classes as dataclass

15 usages
class IO:...
```

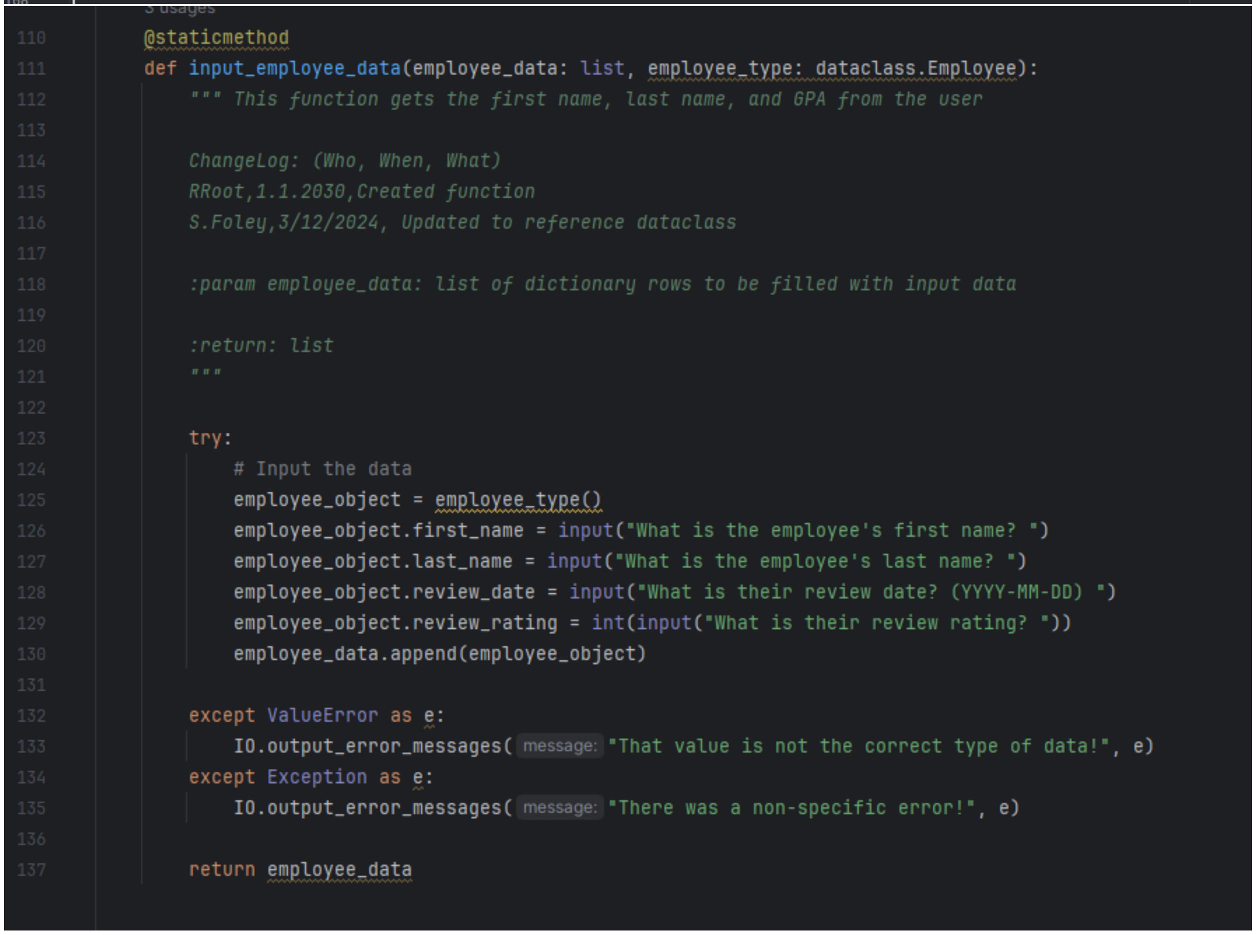
The IO class has the following staticmethods; output_error_messages, output_menu, input_menu_choice, output_employee_data, and input_employee_data.



```
15 usages
class IO:
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    S.Foley,3/12/2024, Updated to reference dataclass
    """
    pass

    6 usages
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...


    1 usage
    @staticmethod
    def output_menu(menu: str):...


    2 usages
    @staticmethod
    def input_menu_choice():...


    2 usages
    @staticmethod
    def output_employee_data(employee_data: list):...
```



```python
    2 usages
    @staticmethod
    def output_employee_data(employee_data: list):
        """ This function displays employee data to the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        S.Foley,3/12/2024, Updated to reference dataclass
        :param employee_data: list of employee object data to be displayed

        :return: None
        """
        message:str = ''
        print()
        print("-" * 50)
        for employee in employee_data:
            if employee.review_rating == 5:
                message = " {} {} is rated as 5 (Leading)"
            elif employee.review_rating == 4:
                message = " {} {} is rated as 4 (Strong)"
            elif employee.review_rating == 3:
                message = " {} {} is rated as 3 (Solid)"
            elif employee.review_rating == 2:
                message = " {} {} is rated as 2 (Building)"
            elif employee.review_rating == 1:
                message = " {} {} is rated as 1 (Not Meeting Expectations"

            print(message.format(*args: employee.first_name, employee.last_name, employee.review_date, employee.review_rating))
        print("-" * 50)
        print()
```

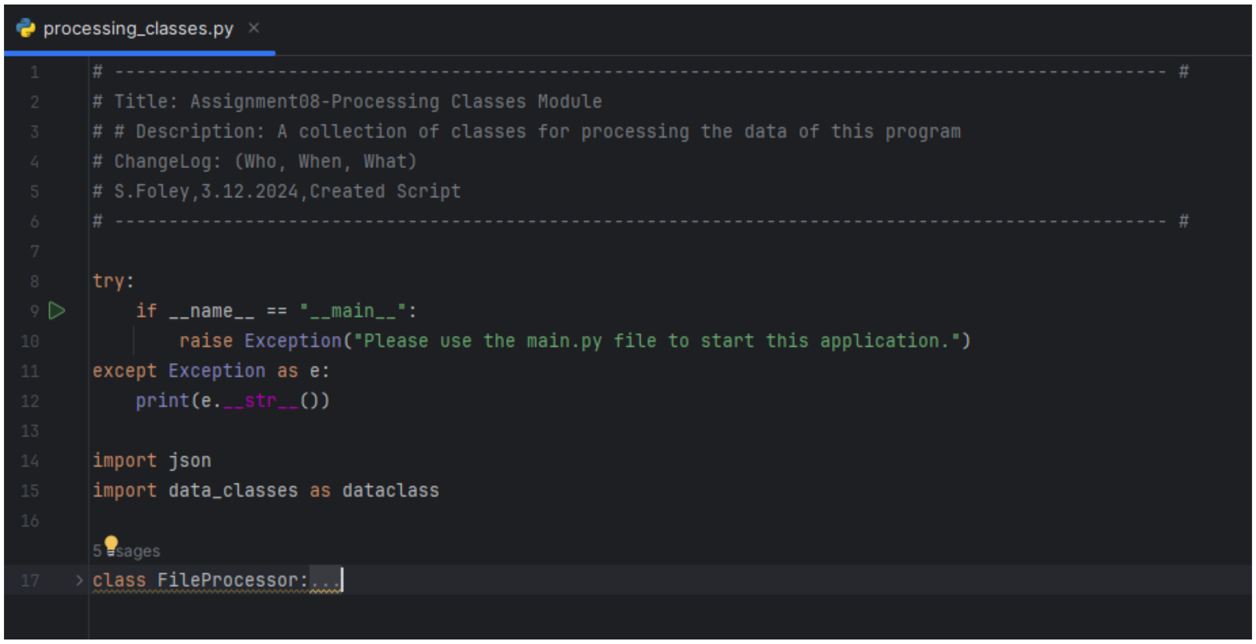


```python
    3 usages
    @staticmethod
    def input_employee_data(employee_data: list, employee_type: dataclass.Employee):
        """ This function gets the first name, last name, and GPA from the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        S.Foley,3/12/2024, Updated to reference dataclass

        :param employee_data: list of dictionary rows to be filled with input data

        :return: list
        """

        try:
            # Input the data
            employee_object = employee_type()
            employee_object.first_name = input("What is the employee's first name? ")
            employee_object.last_name = input("What is the employee's last name? ")
            employee_object.review_date = input("What is their review date? (YYYY-MM-DD) ")
            employee_object.review_rating = int(input("What is their review rating? "))
            employee_data.append(employee_object)

        except ValueError as e:
            IO.output_error_messages(message: "That value is not the correct type of data!", e)
        except Exception as e:
            IO.output_error_messages(message: "There was a non-specific error!", e)

        return employee_data
```

The only changes made from the starter file to this file were the addition of "dataclass." Before Employee.

Processing_classes.py

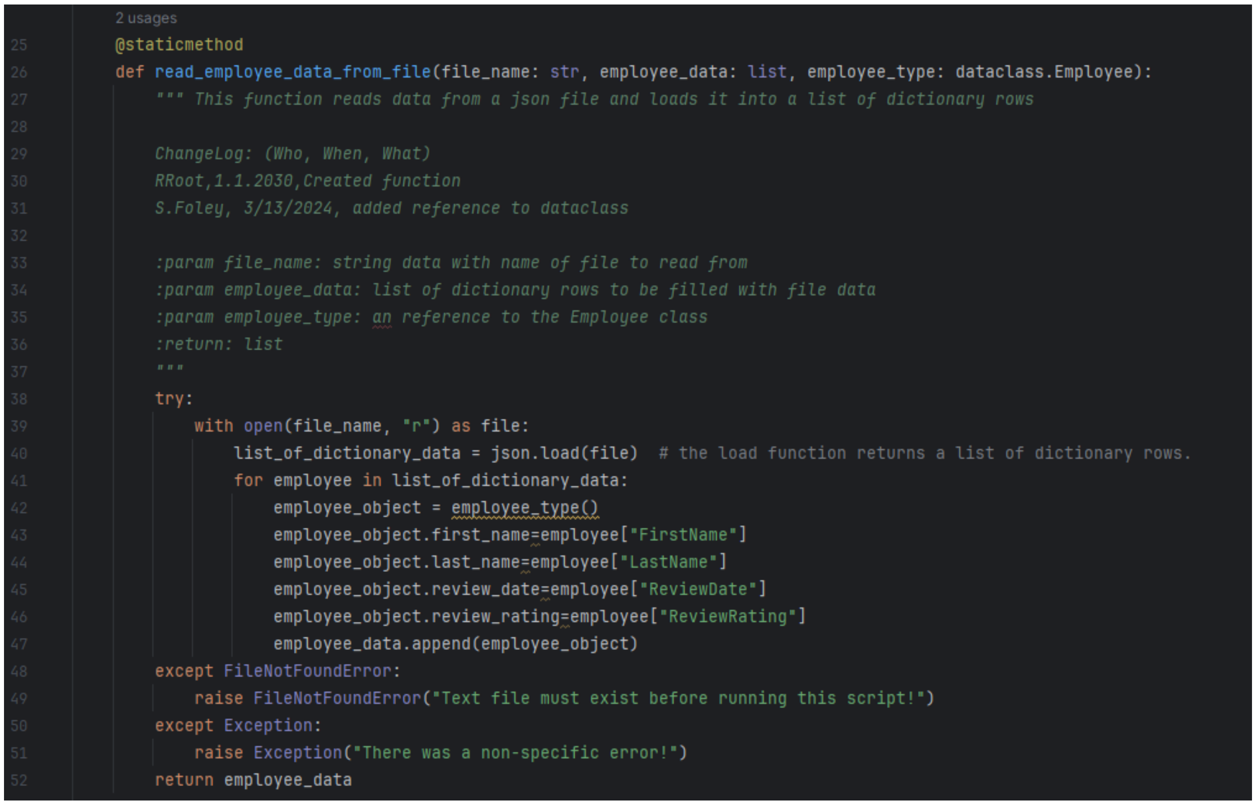
This module covers the Fileprocessor class. The only difference between the previous Fileprocessor version and this current one is that there is an import of the data_classes module at the top, and if this module is attempted to be run, there is an error telling you to run the main.py instead.



processing_classes.py

```python
# ------------------------------------------------------------------------------------------------- #
# Title: Assignment08-Processing Classes Module
# # Description: A collection of classes for processing the data of this program
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# ------------------------------------------------------------------------------------------------- #

try:
    if __name__ == "__main__":
        raise Exception("Please use the main.py file to start this application.")
except Exception as e:
    print(e.__str__())

import json
import data_classes as dataclass

5 usages
class FileProcessor:...
```

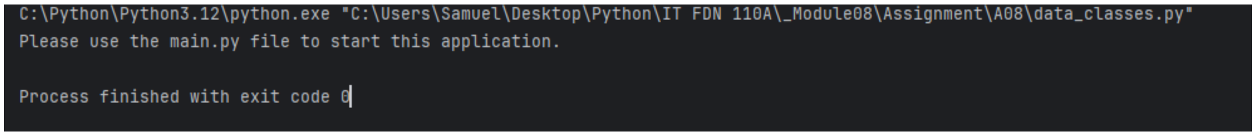
The FileProcessor class houses read_employee_data_from_file, and write_employee_data_to_file. The only other difference from the starter file is that the read data from file class references dataclass.Employee.



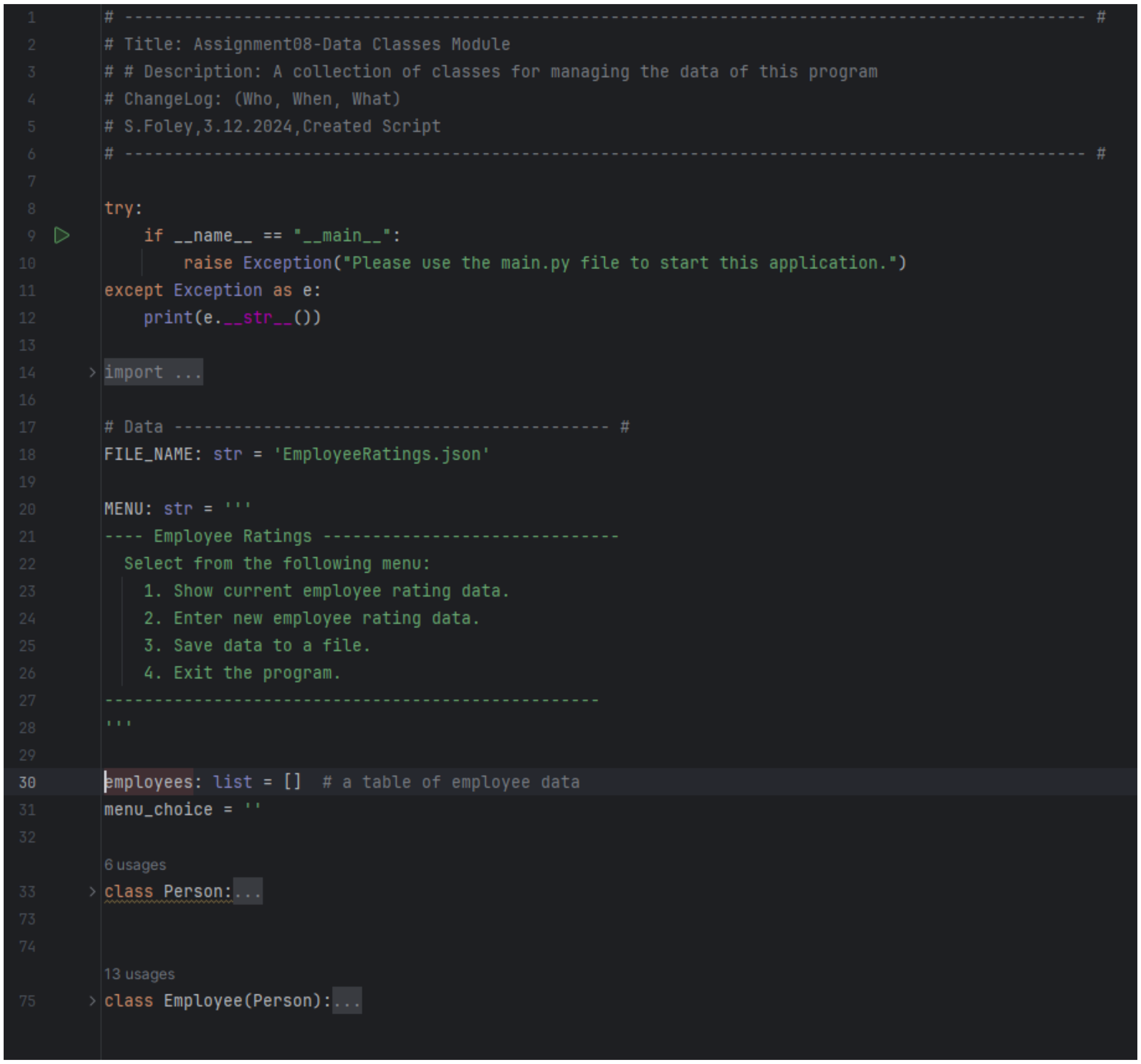
```
    2 usages
    @staticmethod
    def read_employee_data_from_file(file_name: str, employee_data: list, employee_type: dataclass.Employee):
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        S.Foley, 3/13/2024, added reference to dataclass

        :param file_name: string data with name of file to read from
        :param employee_data: list of dictionary rows to be filled with file data
        :param employee_type: an reference to the Employee class
        :return: list
        """
        try:
            with open(file_name, "r") as file:
                list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary rows.
                for employee in list_of_dictionary_data:
                    employee_object = employee_type()
                    employee_object.first_name=employee["FirstName"]
                    employee_object.last_name=employee["LastName"]
                    employee_object.review_date=employee["ReviewDate"]
                    employee_object.review_rating=employee["ReviewRating"]
                    employee_data.append(employee_object)
        except FileNotFoundError:
            raise FileNotFoundError("Text file must exist before running this script!")
        except Exception:
            raise Exception("There was a non-specific error!")
        return employee_data
```

Data_classes.py

This module contains the Person and Employee(Person) classes. Like the other modules not main.py, there is an error that is registered if the module is attempted to be run.



```
C:\Python\Python3.12\python.exe "C:\Users\Samuel\Desktop\Python\IT FDN 110A\_Module08\Assignment\A08\data_classes.py"
Please use the main.py file to start this application.

Process finished with exit code 0
```

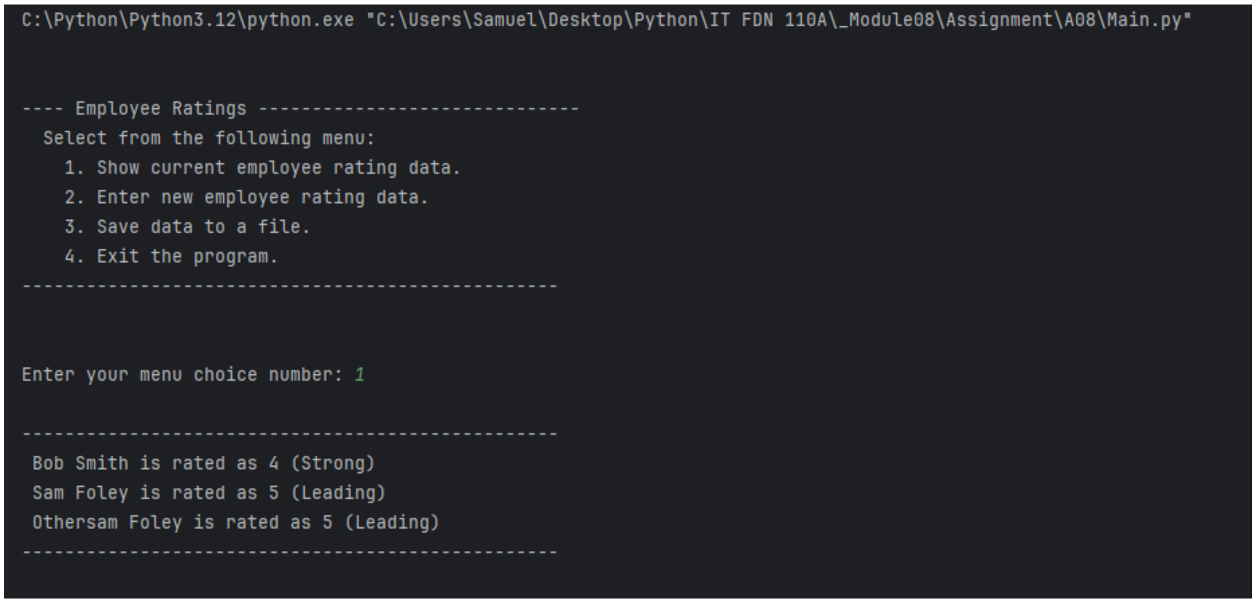


```
# ---------------------------------------------------------------------------------------------------- #
# Title: Assignment08-Data Classes Module
# # Description: A collection of classes for managing the data of this program
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# ---------------------------------------------------------------------------------------------------- #

try:
    if __name__ == "__main__":
        raise Exception("Please use the main.py file to start this application.")
except Exception as e:
    print(e.__str__())

import ...

# Data ---------------------------------------- #
FILE_NAME: str = 'EmployeeRatings.json'

MENU: str = '''
---- Employee Ratings ------------------------------
  Select from the following menu:
    1. Show current employee rating data.
    2. Enter new employee rating data.
    3. Save data to a file.
    4. Exit the program.
--------------------------------------------------
'''

employees: list = []  # a table of employee data
menu_choice = ''

6 usages
class Person:...


13 usages
class Employee(Person):...
```

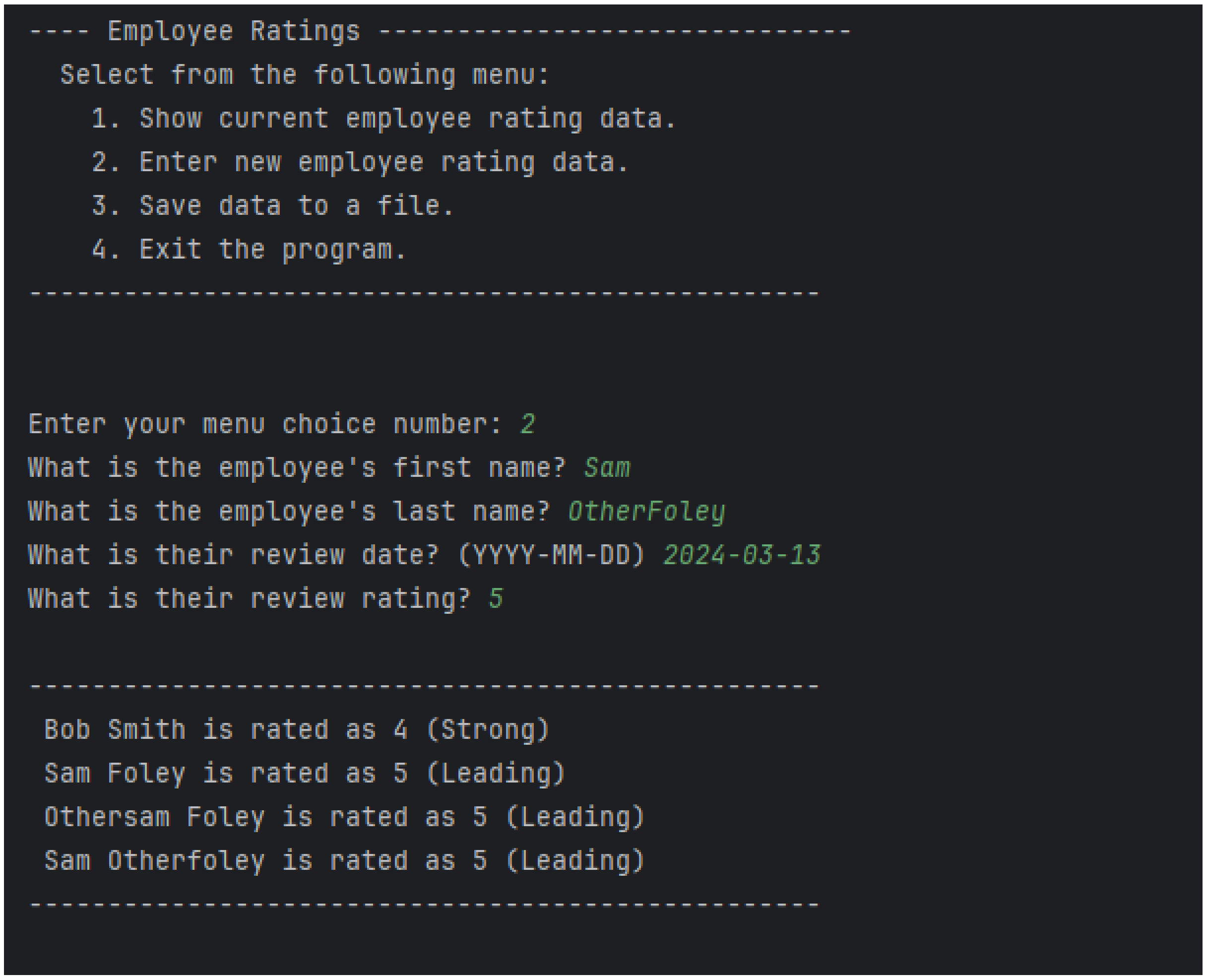
The Person and Employee(Person) classes were not altered from the starter assignment file in any way besides the relocation of the code to this module.

Testing

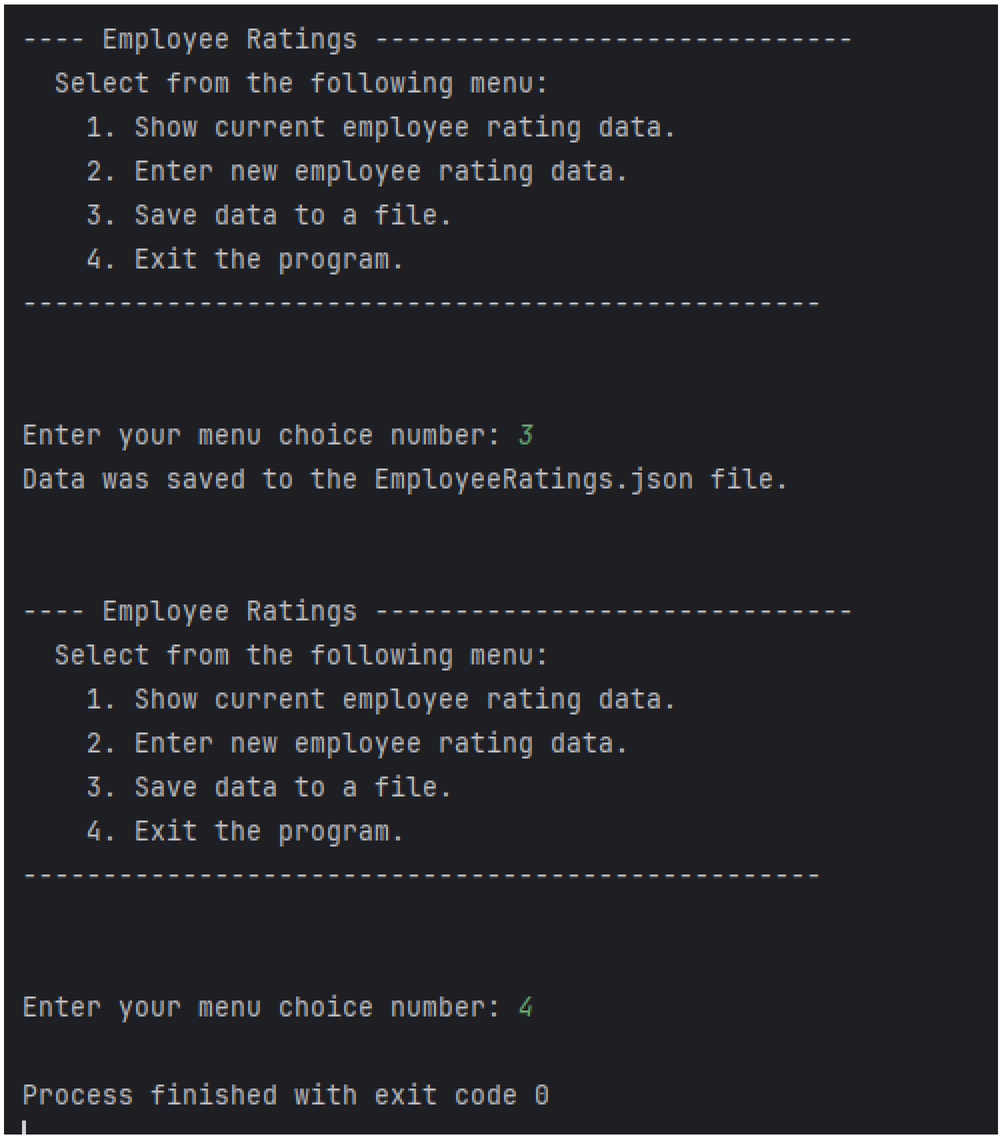
This topic covers the testing of the script to verify that all of the requirements that were laid out were satisfied. The testing will be done using PyCharm and showing the output of both the PyCharm and the json file as shown below:



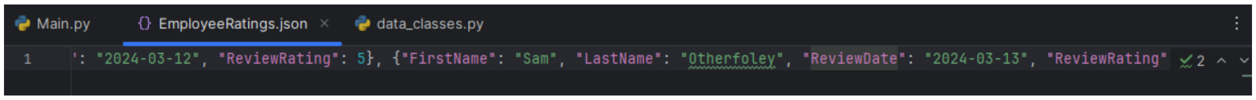
```
C:\Python\Python3.12\python.exe "C:\Users\Samuel\Desktop\Python\IT FDN 110A\_Module08\Assignment\A08\Main.py"


---- Employee Ratings ------------------------------
  Select from the following menu:
    1. Show current employee rating data.
    2. Enter new employee rating data.
    3. Save data to a file.
    4. Exit the program.
--------------------------------------------------


Enter your menu choice number: 1

--------------------------------------------------
 Bob Smith is rated as 4 (Strong)
 Sam Foley is rated as 5 (Leading)
 Othersam Foley is rated as 5 (Leading)
--------------------------------------------------
```



```
---- Employee Ratings ------------------------------
  Select from the following menu:
    1. Show current employee rating data.
    2. Enter new employee rating data.
    3. Save data to a file.
    4. Exit the program.
--------------------------------------------------



Enter your menu choice number: 2
What is the employee's first name? Sam
What is the employee's last name? OtherFoley
What is their review date? (YYYY-MM-DD) 2024-03-13
What is their review rating? 5

--------------------------------------------------
 Bob Smith is rated as 4 (Strong)
 Sam Foley is rated as 5 (Leading)
 Othersam Foley is rated as 5 (Leading)
 Sam Otherfoley is rated as 5 (Leading)
--------------------------------------------------
```



```
---- Employee Ratings ------------------------------
  Select from the following menu:
    1. Show current employee rating data.
    2. Enter new employee rating data.
    3. Save data to a file.
    4. Exit the program.
--------------------------------------------------



Enter your menu choice number: 3
Data was saved to the EmployeeRatings.json file.



---- Employee Ratings ------------------------------
  Select from the following menu:
    1. Show current employee rating data.
    2. Enter new employee rating data.
    3. Save data to a file.
    4. Exit the program.
--------------------------------------------------



Enter your menu choice number: 4

Process finished with exit code 0
```

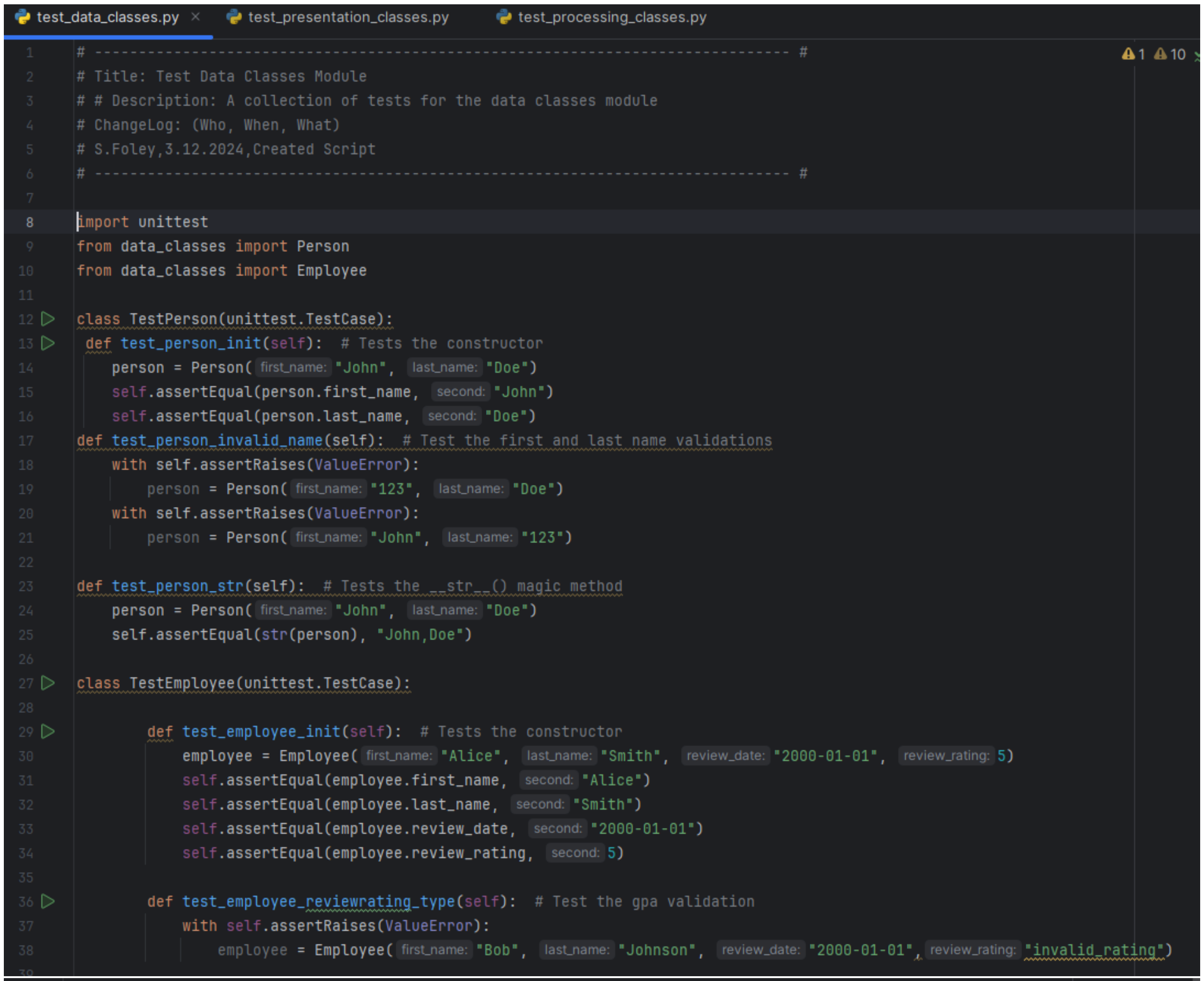


```
': "2024-03-12", "ReviewRating": 5}, {"FirstName": "Sam", "LastName": "Otherfoley", "ReviewDate": "2024-03-13", "ReviewRating"
```

The script was tested in CMD and worked identically to when it was run in Pycharm.

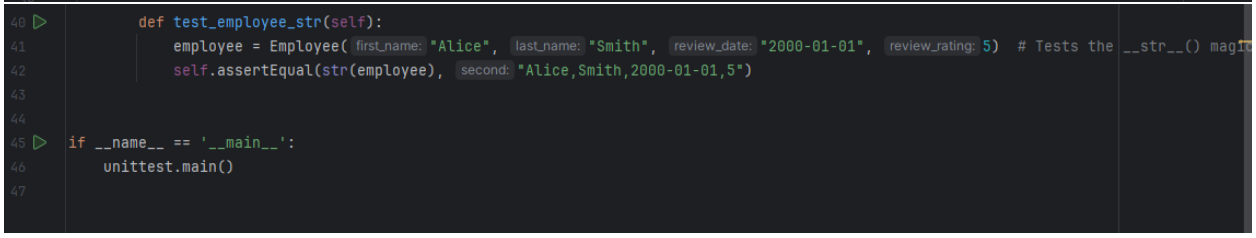
Test_data_classes.py

Below is the test_data_classes that was created to test data_classes Employee and Person.



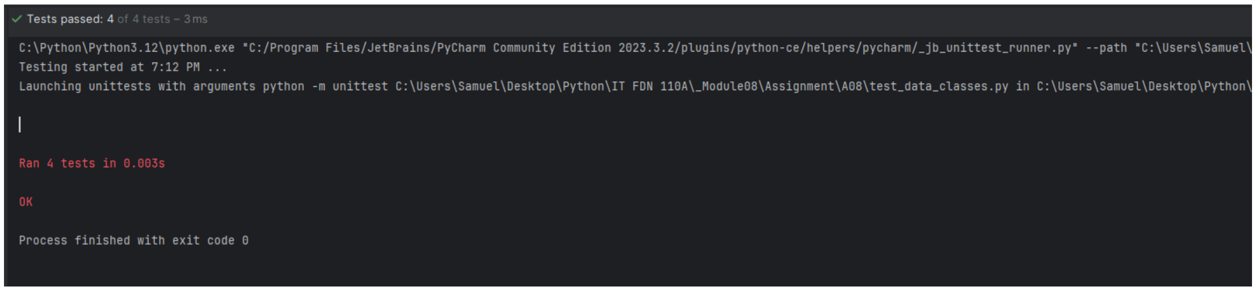
test_data_classes.py × test_presentation_classes.py test_processing_classes.py

```python
# -------------------------------------------------------------------------- #
# Title: Test Data Classes Module
# # Description: A collection of tests for the data classes module
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# -------------------------------------------------------------------------- #

import unittest
from data_classes import Person
from data_classes import Employee

class TestPerson(unittest.TestCase):
 def test_person_init(self):  # Tests the constructor
    person = Person( first_name: "John",  last_name: "Doe")
    self.assertEqual(person.first_name,  second: "John")
    self.assertEqual(person.last_name,  second: "Doe")
def test_person_invalid_name(self):  # Test the first and last name validations
    with self.assertRaises(ValueError):
        person = Person( first_name: "123",  last_name: "Doe")
    with self.assertRaises(ValueError):
        person = Person( first_name: "John",  last_name: "123")

def test_person_str(self):  # Tests the __str__() magic method
    person = Person( first_name: "John",  last_name: "Doe")
    self.assertEqual(str(person), "John,Doe")

class TestEmployee(unittest.TestCase):

        def test_employee_init(self):  # Tests the constructor
            employee = Employee( first_name: "Alice",  last_name: "Smith",  review_date: "2000-01-01",  review_rating: 5)
            self.assertEqual(employee.first_name,  second: "Alice")
            self.assertEqual(employee.last_name,  second: "Smith")
            self.assertEqual(employee.review_date,  second: "2000-01-01")
            self.assertEqual(employee.review_rating,  second: 5)

        def test_employee_reviewrating_type(self):  # Test the gpa validation
            with self.assertRaises(ValueError):
                employee = Employee( first_name: "Bob",  last_name: "Johnson",  review_date: "2000-01-01", review_rating: "invalid_rating")
```



```python
        def test_employee_str(self):
            employee = Employee( first_name: "Alice",  last_name: "Smith",  review_date: "2000-01-01",  review_rating: 5)  # Tests the __str__() magi
            self.assertEqual(str(employee),  second: "Alice,Smith,2000-01-01,5")


if __name__ == '__main__':
    unittest.main()
```

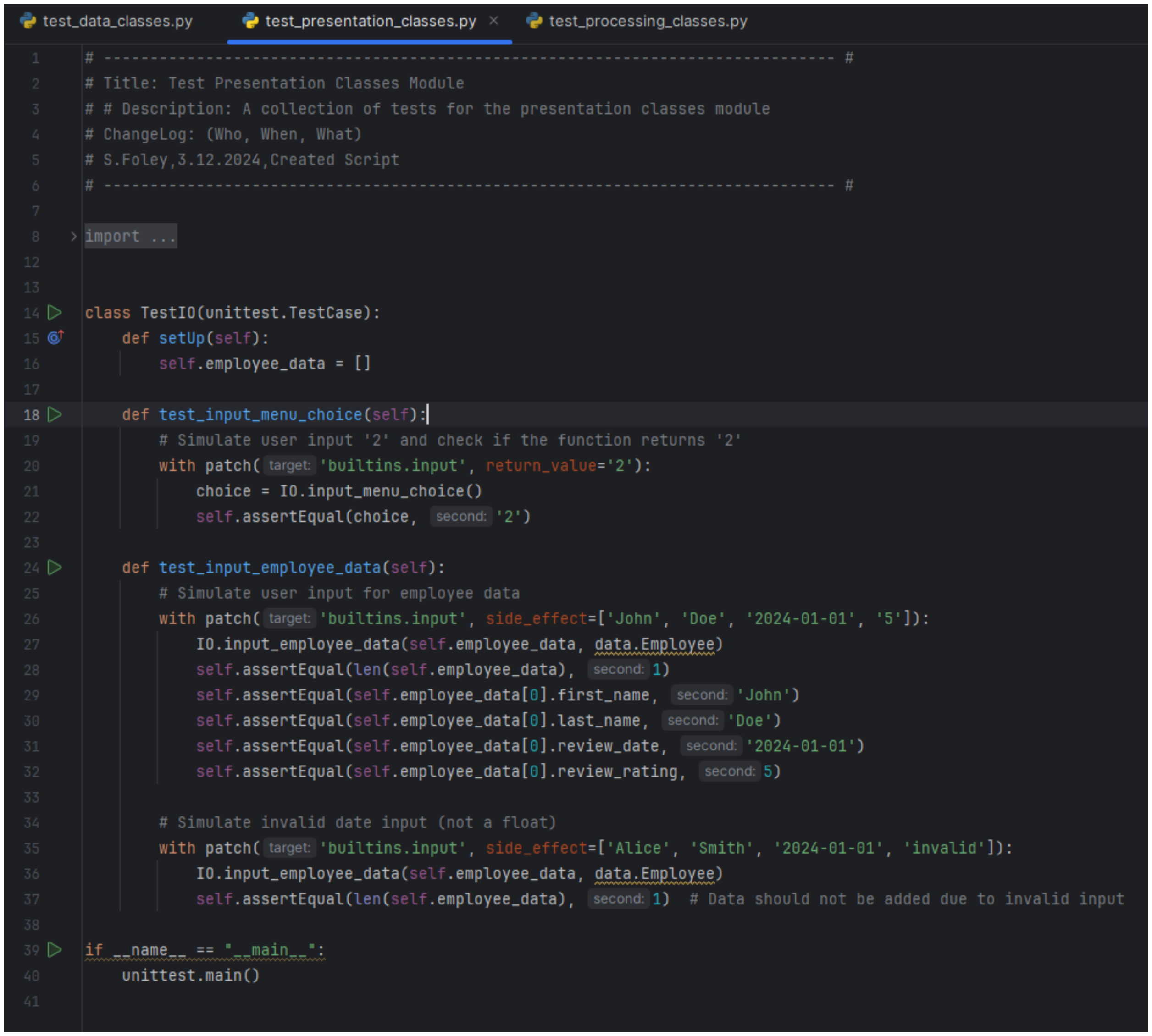
I ran the test for the following results:



```
✓ Tests passed: 4 of 4 tests – 3 ms

C:\Python\Python3.12\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.3.2/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py" --path "C:\Users\Samuel\
Testing started at 7:12 PM ...
Launching unittests with arguments python -m unittest C:\Users\Samuel\Desktop\Python\IT FDN 110A\_Module08\Assignment\A08\test_data_classes.py in C:\Users\Samuel\Desktop\Python\

|

Ran 4 tests in 0.003s

OK

Process finished with exit code 0
```

Test_presentation_classes.py

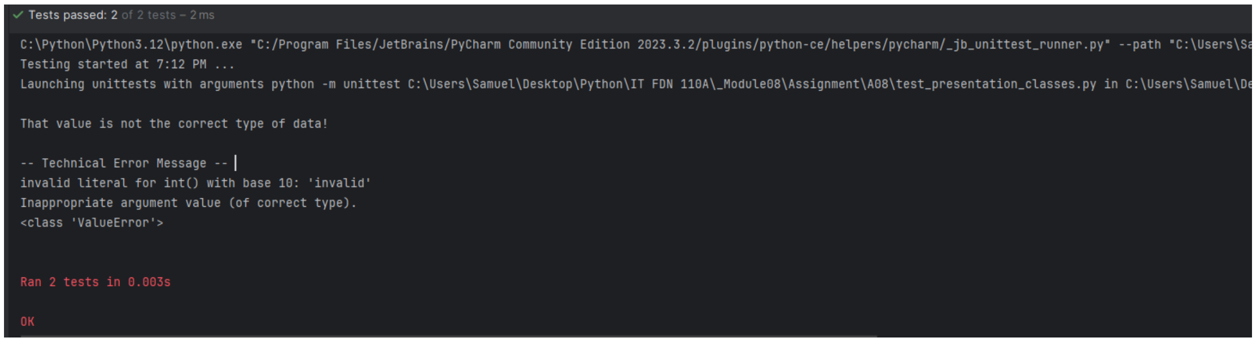
Below is the test_presentation_classes that was created to test the IO class.



test_data_classes.py test_presentation_classes.py test_processing_classes.py

```python
# -------------------------------------------------------------------------------- #
# Title: Test Presentation Classes Module
# # Description: A collection of tests for the presentation classes module
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# -------------------------------------------------------------------------------- #

import ...


class TestIO(unittest.TestCase):
    def setUp(self):
        self.employee_data = []

    def test_input_menu_choice(self):
        # Simulate user input '2' and check if the function returns '2'
        with patch( target: 'builtins.input', return_value='2'):
            choice = IO.input_menu_choice()
            self.assertEqual(choice,  second: '2')

    def test_input_employee_data(self):
        # Simulate user input for employee data
        with patch( target: 'builtins.input', side_effect=['John', 'Doe', '2024-01-01', '5']):
            IO.input_employee_data(self.employee_data, data.Employee)
            self.assertEqual(len(self.employee_data),  second: 1)
            self.assertEqual(self.employee_data[0].first_name,  second: 'John')
            self.assertEqual(self.employee_data[0].last_name,  second: 'Doe')
            self.assertEqual(self.employee_data[0].review_date,  second: '2024-01-01')
            self.assertEqual(self.employee_data[0].review_rating,  second: 5)

        # Simulate invalid date input (not a float)
        with patch( target: 'builtins.input', side_effect=['Alice', 'Smith', '2024-01-01', 'invalid']):
            IO.input_employee_data(self.employee_data, data.Employee)
            self.assertEqual(len(self.employee_data),  second: 1)  # Data should not be added due to invalid input

if __name__ == "__main__":
    unittest.main()

```

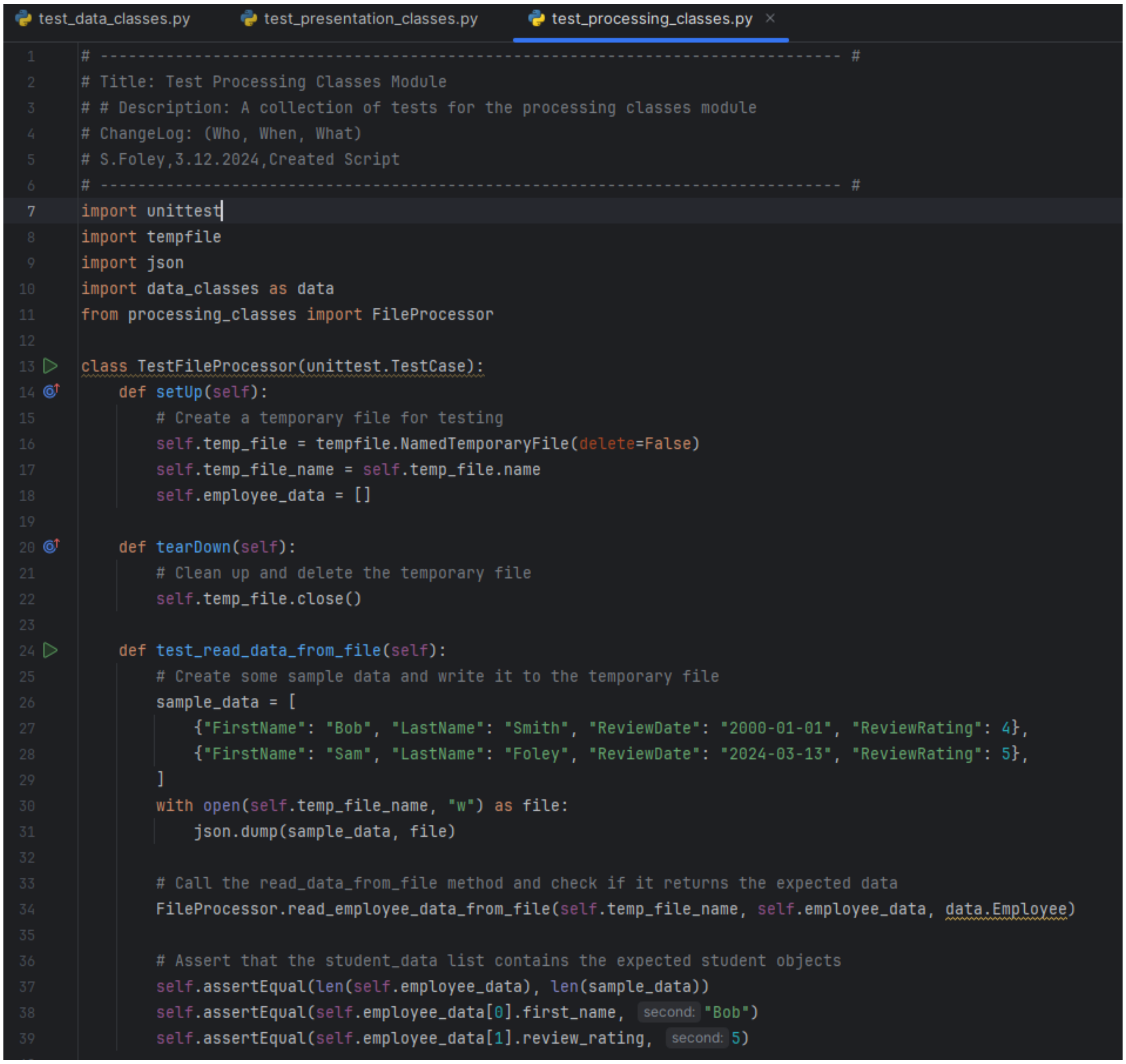
I ran the test for the following results:



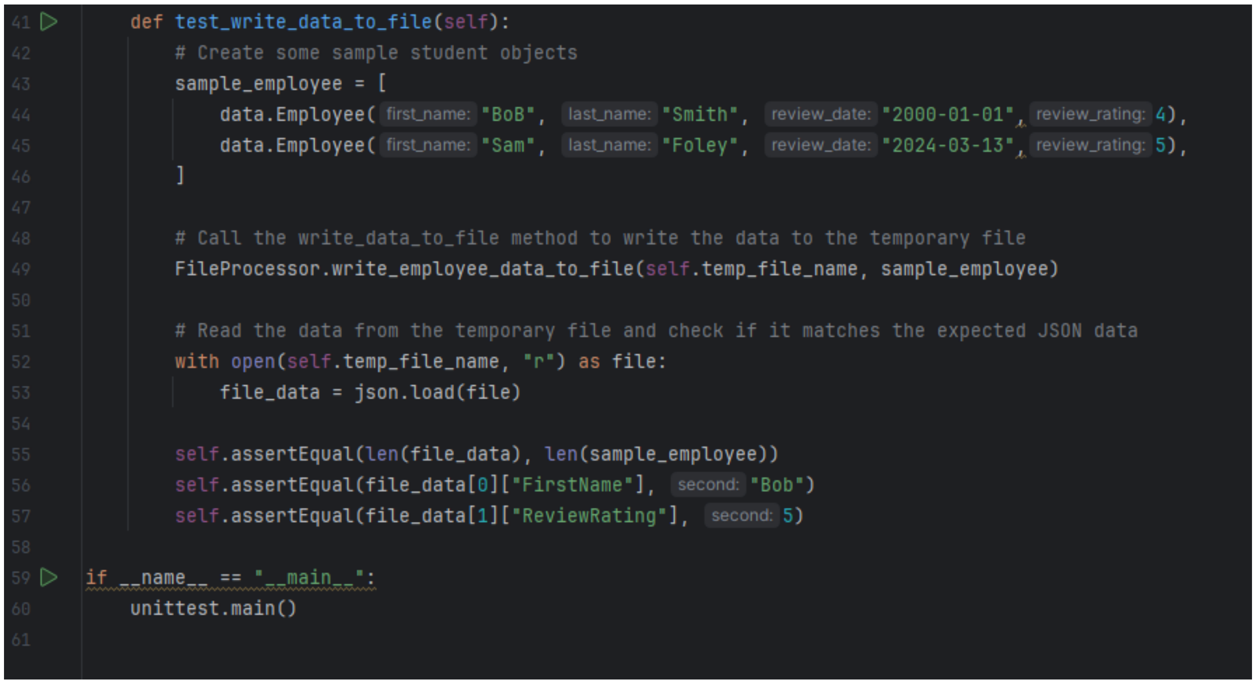
```
✓ Tests passed: 2 of 2 tests – 2 ms
C:\Python\Python3.12\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.3.2/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py" --path "C:\Users\Sa
Testing started at 7:12 PM ...
Launching unittests with arguments python -m unittest C:\Users\Samuel\Desktop\Python\IT FDN 110A\_Module08\Assignment\A08\test_presentation_classes.py in C:\Users\Samuel\De

That value is not the correct type of data!

-- Technical Error Message --
invalid literal for int() with base 10: 'invalid'
Inappropriate argument value (of correct type).
<class 'ValueError'>


Ran 2 tests in 0.003s

OK
```

Test_processing_classes.py

Below is the test_processing_classes that was created to test the read and write to file classes.

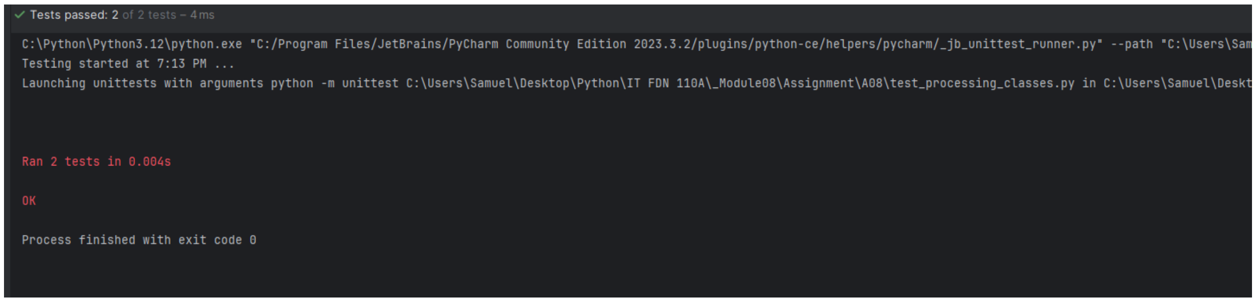


```python
# ------------------------------------------------------------------------------- #
# Title: Test Processing Classes Module
# # Description: A collection of tests for the processing classes module
# ChangeLog: (Who, When, What)
# S.Foley,3.12.2024,Created Script
# ------------------------------------------------------------------------------- #
import unittest
import tempfile
import json
import data_classes as data
from processing_classes import FileProcessor

class TestFileProcessor(unittest.TestCase):
    def setUp(self):
        # Create a temporary file for testing
        self.temp_file = tempfile.NamedTemporaryFile(delete=False)
        self.temp_file_name = self.temp_file.name
        self.employee_data = []

    def tearDown(self):
        # Clean up and delete the temporary file
        self.temp_file.close()

    def test_read_data_from_file(self):
        # Create some sample data and write it to the temporary file
        sample_data = [
            {"FirstName": "Bob", "LastName": "Smith", "ReviewDate": "2000-01-01", "ReviewRating": 4},
            {"FirstName": "Sam", "LastName": "Foley", "ReviewDate": "2024-03-13", "ReviewRating": 5},
        ]
        with open(self.temp_file_name, "w") as file:
            json.dump(sample_data, file)

        # Call the read_data_from_file method and check if it returns the expected data
        FileProcessor.read_employee_data_from_file(self.temp_file_name, self.employee_data, data.Employee)

        # Assert that the student_data list contains the expected student objects
        self.assertEqual(len(self.employee_data), len(sample_data))
        self.assertEqual(self.employee_data[0].first_name, second: "Bob")
        self.assertEqual(self.employee_data[1].review_rating, second: 5)
```



```python
    def test_write_data_to_file(self):
        # Create some sample student objects
        sample_employee = [
            data.Employee( first_name: "BoB",  last_name: "Smith",  review_date: "2000-01-01", review_rating: 4),
            data.Employee( first_name: "Sam",  last_name: "Foley",  review_date: "2024-03-13", review_rating: 5),
        ]

        # Call the write_data_to_file method to write the data to the temporary file
        FileProcessor.write_employee_data_to_file(self.temp_file_name, sample_employee)

        # Read the data from the temporary file and check if it matches the expected JSON data
        with open(self.temp_file_name, "r") as file:
            file_data = json.load(file)

        self.assertEqual(len(file_data), len(sample_employee))
        self.assertEqual(file_data[0]["FirstName"],  second: "Bob")
        self.assertEqual(file_data[1]["ReviewRating"],  second: 5)

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

I ran the test for the following results:



```
✓ Tests passed: 2 of 2 tests – 4 ms
C:\Python\Python3.12\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2023.3.2/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py" --path "C:\Users\Sam
Testing started at 7:13 PM ...
Launching unittests with arguments python -m unittest C:\Users\Samuel\Desktop\Python\IT FDN 110A\_Module08\Assignment\A08\test_processing_classes.py in C:\Users\Samuel\Deskt



Ran 2 tests in 0.004s

OK

Process finished with exit code 0
```

Summary

This document described the steps I took to write my first script in PyCharm for Assignment 08.