Name: Sohail Nassiri

Date: September 25th, 2024

Course: IT FDN 110 B Su 24 - Foundations of Programming: Python

GitHub URL: https://github.com/jnts221/IntroToProg-Python-Mod08

Assignment 08 – Creating a Python Application Using Multiple Modules and Unit Testing

Introduction

This assignment requires the use of constants, variables, and a variety of functions, in a Python script. In addition, it incorporates the use of string formatting, while and for loops, programming menus, conditional logic, classes, object-oriented programming, exception handling, java script-oriented notation (JSON) files to work with data, multiple modules and unit testing. The purpose of the Python script is to allow users to enter and display multiple review ratings for employees and their review date along with the ability to read and save the data to/from a JSON file if desired.

Creating the Python Script

Using the PyCharm IDE, a Python script is created for each module and unit test. The code begins with the script header, as shown in **Figure 1a-1g**, in order to give some background information on the purpose of each script. In this case, the scripts are being written in order to complete the requirements of Assignment 8 demonstrating the use of important concepts in Module 08 being:

- Modules
- Unit Testing

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment08
# # Description: A collection of classes for managing the application
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------------------ #
```

Figure 1a. Script Header for main.py

```
# ------------------------------------------------------------------------------------------ #
# Title: Processing Classes Module
# # Description: A collection of processing classes for managing the application
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------------------ #
```

Figure 1b. Script Header for processing_classes.py

```
# ------------------------------------------------------------------------------- #
# Title: Presentation Classes Module
# # Description: A collection of presentation classes for managing the application
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------- #
```

Figure 1c. Script Header for presentation_classes.py

```
# ------------------------------------------------------------------------------- #
# Title: Data Classes Module
# # Description: A collection of data classes for managing the application
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------- #
```

Figure 1d. Script Header for data_classes.py

```
# ------------------------------------------------------------------------------- #
# Title: Test Processing Classes Module
# # Description: A collection of tests for the processing classes module
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------- #
```

Figure 1e. Script Header for test_processing_classes.py

```
# ------------------------------------------------------------------------------- #
# Title: Test Presentation Classes Module
# # Description: A collection of tests for the presentation classes module
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------- #
```

Figure 1f. Script Header for test_presentation_class.py

```
# # Description: A collection of tests for the data classes module
# ChangeLog: (Who, When, What)
# Sohail Nassiri,09.25.2024,Created Script
# ------------------------------------------------------------------------------- #
```

Figure 1g. Script Header for test_data_classes.py

The main.py file is used to start the application and the code first begins by a try-except block and if-else statement to let the user know that is the case and that it should not be imported into the other modules. Next, the FileProcessor method and read_employee_data_from_file function from the processing_classes module is used to read data from the "EmployeeRatings.json" file into a list of objects. Then, a while loop is used to present the choice menu to the user using the IO method and output_menu and menu_choice functions from the presentation_classes module. An if-else statement is used for the menu choice selections as follows:

- Selection 1 (Displays the current data) – The IO method and output_employee_data function from the presentation_classes module is used to output the current employee data.
- Selection 2 (Get new data and display) – The IO method and input_employee_data function from the presentation_classes module is used to receive and store user input and the output_employee_data function is used to display the data that was entered.
- Selection 3 (Save data in a file) – The FileProcessor method and write_employee_data_to_file function from the processing_classes module is used to write and save the current data to the .json file.
- Selection 4 (End the program) – A break is used to exit the while loop and end the program.

All of the code as described above for main.py is shown in **Figure 2**.

```
try:
    if __name__ == "__main__":
        import processing_classes as proc
        import presentation_classes as pres
        import data_classes as data
    else:
        raise Exception("This file starts the application and should not be imported.")
except Exception as e:
    print(e.__str__())


# Beginning of the main body of this script
employees = proc.FileProcessor.read_employee_data_from_file(file_name=data.FILE_NAME, employee_data=data.employees,
                                                            employee_type=data.Employee)

# Repeat the follow tasks
while True:
    pres.IO.output_menu(menu=data.MENU)
    menu_choice = pres.IO.input_menu_choice()

    if menu_choice == "1":  # Display current data
        pres.IO.output_employee_data(employee_data=employees)
        continue

    elif menu_choice == "2":  # Get new data (and display the change)
        employees = pres.IO.input_employee_data(employee_data=employees, employee_type=data.Employee)
        pres.IO.output_employee_data(employee_data=employees)
        continue

    elif menu_choice == "3":  # Save data in a file
        proc.FileProcessor.write_employee_data_to_file(file_name=data.FILE_NAME, employee_data=employees)
        pres.IO.output_employee_data(employee_data=employees)
        continue

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

Figure 2. Body Code for main.py

The processing_classes.py file is imported into the main.py file as a module. First, the code begins by a try-except block and if-else statement to let the user know it should not be run to the start application. Next, the FileProcessor (a collection of processing layer functions that work with json files) class is created with the following methods and functions:

- read_employee_data_from_file with parameters file_name (str), employee_data (list) and employee_type (object). The function reads data from a .json file using the open("r" – read) function and loads (with the json.load function) it into a list of employee objects. The first_name, last_name, review_data, and review_rating parameters are created for the list of employee objects and appended to employee_data. Error handling/exceptions are incorporated in case the file is not found or another general error occurs.
- write_employee_data_to_file with parameters file_name (str), employee_data (list). The function writes data to the .json file using the open("w" – read) function and adds/saves (with the json.dump function) it. In order to convert the data to an acceptable format for json, the employee objects are converted to a list of dictionary data. Error handling/exceptions are incorporated in case there is an error writing to the .json file (such as incorrect format) or another general error occurs.

All of the code as described above for processing_classes.py is shown in **Figure 3a-3b**.

```python
try:
    if __name__ == "__main__":
        raise Exception("Please use the main.py file to start this application.")
    else:
        import json
        import data_classes as data
        import presentation_classes as pres
except Exception as e:
    print(e.__str__())


class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.25.2024,Created Class and converted code to use employee objects instead of dictionaries
    """

    2 usages
    @staticmethod
    def read_employee_data_from_file(file_name: str, employee_data: list, employee_type: object):
        """ This function reads data from a json file and loads it into a list of employee objects

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.25.2024,Created function and converted list of dictionaries to list of employee objects

        :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: list of employee objects to be filled with file data

        :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(first_name=employee["FirstName"],
                                                    last_name=employee["LastName"],
                                                    review_date=employee["ReviewDate"],
                                                    review_rating=employee["ReviewRating"])
```

Figure 3a. Body Code for processing_classes.py

```
                employee_data.append(employee_object)
        except FileNotFoundError as e:
            pres.IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            pres.IO.output_error_messages( message: "There was a non-specific error!", e)
        return employee_data

    2 usages
    @staticmethod
    def write_employee_data_to_file(file_name: str, employee_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.25.2024,Created function and converted code to use employee objects instead of dictionaries

        :param file_name: string data with name of file to write to
        :param employee_data: list of dictionary rows to be writen to the file

        :return: None
        """
        try:
            list_of_dictionary_data: list = []
            for employee in employee_data:  # Convert List of Employee objects to list of dictionary rows.
                employee_json: dict = {"FirstName": employee.first_name,
                                       "LastName": employee.last_name,
                                       "ReviewDate": str(employee.review_date),
                                       "ReviewRating": employee.review_rating}
                list_of_dictionary_data.append(employee_json)

            with open(file_name, "w") as file:
                json.dump(list_of_dictionary_data, file)
            print("The following data has been saved to the file:")
        except TypeError as e:
            pres.IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
        except Exception as e:
            pres.IO.output_error_messages( message: "There was a non-specific error!", e)
```

Figure 3b. Body Code for processing_classes.py

The presentation_classes.py file is imported into the main.py file as a module. First, the code begins by a try-except block and if-else statement to let the user know it should not be run to the start application. Next, the IO class (a collection of presentation layer functions that manage user input and output) is created with the following methods and functions:

- output_error_messages with parameters message (str), error (Exception = None) and employee_type. The function displays custom error messages to the user.
- output_menu with parameter menu (str) which displays the menu of choices to the user.
- Input_menu_choice which gets a menu choice from user. A try-except block and if statement is used to receive the menu choice selection from the user. If an invalid selection is made, it raises an Exception to let the user know they must choose from the selections on the menu (#1-4). Lastly, the choice is returned.
- output_employee_data with parameters employee_data (list) displays the review ratings to the user as follows:
    - 1 – Does Not Meet Expectations

    - 2 – Meets Some Expectations
    - 3 – Meets Expectations
    - 4 – Exceeds Expectations
    - 5 – Far Exceeds Expectations
- input_employee_data with parameters employee_data (list) and employee_type (object) which gets the first name, last name, review date and review rating from the user. The inputted data is then appended to employee_data. Exception handling is incorporated in case the user enters the incorrect type of data such as a float for the review rating or non-alphabetical character for the first and last name as well as any other general errors.

All of the code as describe above for presentation_classes.py is shown in **Figure 4a-d**.

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


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

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.25.2024,Created Class, added menu output and input functions,added a function to display the data,
    added a function to display custom error message, and converted methods to use student objects instead of
    dictionaries
    """
    pass

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays custom error messages to the user

        ChangeLog: (Who, When, What)
        Sohail Nassiri,09.25.2024,Created function

        :param message: string with message data to display
        :param error: Exception object with technical message to display

        :return: None
        """

        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')
```

Figure 4a. Body Code for presentation_classes.py

```python
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

    ChangeLog: (Who, When, What)
     Sohail Nassiri,09.25.2024,Created function

    :return: None
    """
    print()
    print(menu)
    print()

@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.25.2024,Created function

    :return: string with the users choice
    """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1", "2", "3", "4"):  # Note these are strings
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())  # passing the exception object to avoid the technical message

    return choice
```

Figure 4b. Body Code for presentation_classes.py

```
3 usages
@staticmethod
def output_employee_data(employee_data: list):
    """ This function displays the review ratings to the user

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.25.2024,Created function and converted code to use student objects instead of dictionaries

    :param employee_data: list of student object data to be displayed

    :return: None
    """
    print()
    print("-" * 50)
    message = ""
    for employee in employee_data:
        if employee.review_rating == 1:
            message = " On {}, {} {} earned a rating of {} (Does Not Meet Expectations)."
        elif employee.review_rating == 2:
            message = " On {}, {} {} earned a rating of {} (Meets Some Expectations)."
        elif employee.review_rating == 3:
            message = " On {}, {} {} earned a rating of {} (Meets Expectations)."
        elif employee.review_rating == 4:
            message = " On {}, {} {} earned a rating of {} (Exceeds Expectations)."
        elif employee.review_rating == 5:
            message = " On {}, {} {} earned a rating of {} (Far Exceeds Expectations)."
        print(message.format( *args: employee.review_date, employee.first_name, employee.last_name,
                             employee.review_rating))

    print("-" * 50)
    print()
```

Figure 4c. Body Code for presentation_classes.py

```
4 usages
@staticmethod
def input_employee_data(employee_data: list, employee_type: object):
    """ This function gets the first name, last name, review date, and review rating from the user

    ChangeLog: (Who, When, What)
     Sohail Nassiri,09.25.2024,Created function and converted code to use student objects instead of dictionaries

    :param employee_data: list of dictionary rows to be filled with input data
    :param employee_type: list of employee objects to be filled with file data

    :return: list
    """

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

    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
```

Figure 4d. Body Code for presentation_classes.py

The data_classes.py file is imported into the main.py file as a module. First, the code begins by a try-except block and if-else statement to let the user know it should not be run to the start application. Next, the data constants are defined, which are designated in upper case. Also, type hints are used to allow the reader to know the data type associated with each constant or variable. The following constants are assigned:

- The employee ratings program menu (see below), is assigned to MENU (string).
  ---- 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.

- The name of the .json file, 'EmployeeRatings.json', is assigned to FILE_NAME (string)

The global data variables, which are declared outside of the functions and accessible anywhere within the script, are defined and set to empty strings as follows:

- employees (list) is set to an empty list
- menu_choice (string) is set to empty string

Then, the following four classes are created in order to organize the code using object-oriented programming (see **Figures 5a-5d**):

- Person – a class representing person data
    - The class contains the properties first_name (str) and last_name (str) to represent each person's first and last name.
    - A constructor, which is a way to create objects with an initial set of values, is created using __init__ with private attributes for first_name and last_name. This is also known as an instance method.
    - Properties are used to get and set the instance variables. In this case, its use is for additional type checking and logic.
    - The setter contains the validation check to ensure the person's first and last name entered is alphabetical. During this check, the first_name and last_name variables are secret (denoted by the "self.__") and cannot be accessed outside of the class.
    - Ultimately, the constructor inside the Person class jumps to the setters and runs the desired validation steps prior to assigning the user's input to the first_name and last_name variables.
- Employee – a class representing employee data
    - The class contains the properties first_name (str), last_name (str), review_date (str) and review_rating (int). Also, it inherits code from the Person class, known as inheritance. Inheritance allows for increased flexibility and more concise code since it prevents the need to duplicate information between the Person and Employee classes.
    - The super().__init__ constructor is used to pass the parameter data to the Person "super" or "parent" class. Thus, the logic setters used in the Person class do not need to be duplicated in the Employee class since they are already occurring.
    - The same approach of constructors, properties, and setters are used for the review_date and review_rating. The constructor inside the Employee class jumps to the setters prior to assigning the user's input to review_date and review_rating. In addition, validation for the user's input is incorporated for review_data by ensuring it is in ISO 8601 format and for review_rating by ensuring it is an integer value between 1-5.
    - The __str__() method is overridden to return the full employee data being the first name, last name, review date and review rating.

All of the code as described above for data_classes.py is shown in **Figure 5a-5d**.

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

# 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 = ''
```

Figure 5a. Body Code for data_classes.py

```python
class Person:
    """
    A class representing person data.

    Properties:
        first_name (str): The employee's first name.
        last_name (str): The employee's last name.

    ChangeLog:
    Sohail Nassiri,09.25.2024: Created the class.
    """

    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        return self.__first_name.title()  # formatting code

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__first_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    @property
    def last_name(self):
        return self.__last_name.title()  # formatting code

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    def __str__(self):
        return f'{self.first_name},{self.last_name}'
```

Figure 5b. Body Code for data_classes.py

```python
class Employee(Person):
    """
    A class representing employee data.

    Properties:
        first_name (str): The employee's first name.
        last_name (str): The employee's last name.
        review_date (str): The employee's review date.
        review_rating (int): The review rating of the employee.

    ChangeLog: (Who, When, What)
    Sohail Nassiri,09.25.2024,Created Class, added properties and private attribute, moved first_name and last_name into
    a parent class
    """

    def __init__(self, first_name: str = '', last_name: str = '', review_date: str = '1900-01-01',
                 review_rating: int = 3):
        super().__init__(first_name=first_name, last_name=last_name)
        self.review_date = review_date
        self.review_rating = review_rating

    11 usages (5 dynamic)
    @property
    def review_date(self):
        return self.__review_date

    9 usages (5 dynamic)
    @review_date.setter
    def review_date(self, value: str):
        try:  # using a try block to capture when an input cannot be changed to format other than ISO 8601
            date.fromisoformat(value)
            self.__review_date = value

        except ValueError:
            raise ValueError("Review date must be in ISO 8601 format (YYYY-MM-DD).")
```

Figure 5c. Body Code for data_classes.py

```python
    @property
    def review_rating(self):
        return self.__review_rating

    @review_rating.setter
    def review_rating(self, value: int):
        try:  # using a try block to capture when an input cannot be changed to a float
            self.__review_rating = value
            if value not in (1, 2, 3, 4, 5):
                raise ValueError("Review rating must be between 1 and 5.")
        except ValueError:
            raise ValueError("Review rating must be a numeric integer value between 1 - 5.")

    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.review_date},{self.review_rating}'
```

Figure 5d. Body Code for data_classes.py

Unit Tests

The test_processing_classes.py file consists of two tests for the processing classes module. First, a temporary file is set up for testing and then cleaned up and deleted (tore down) after completion of tests. The two tests carried out are created as follows:

- test_read_data_from_file – sample data is created and written to the temporary file. The read_data_from_file method is called to check if the data is properly read. The assertEqual function is used to assert the employee_data list contains the expected employee objects.
- test_write_data_to_file – Again, sample data is created and written to the temporary file. The write_data_to_file method is called to write the data to the temporary file. The data is then read from the temporary file to check if it matched the expected JSON data.

All of the code as described above for test_processing_classes.py and the test results are shown in **Figure 6a-6d.**

```python
import unittest
import tempfile
import json
import data_classes as data
import processing_classes as proc


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": "John", "LastName": "Doe", "ReviewDate": "2024-03-22", "ReviewRating": 3},
            {"FirstName": "Alice", "LastName": "Smith", "ReviewDate": "2024-04-03", "ReviewRating": 5},
        ]
```

Figure 6a. Body Code for test_processing_classes.py

```
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
proc.FileProcessor.read_employee_data_from_file(self.temp_file_name, self.employee_data, data.Employee)

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

f test_write_data_to_file(self):
# Create some sample employee objects
sample_employees = [
    data.Employee( first_name: "John", last_name: "Doe", review_date: "2024-03-22", review_rating: 3),
    data.Employee( first_name: "Alice", last_name: "Smith", review_date: "2024-04-03", review_rating: 5),
]

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

Figure 6b. Body Code for test_processing_classes.py

```
        # 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_employees))
        self.assertEqual(file_data[0]["FirstName"], second: "John")
        self.assertEqual(file_data[1]["ReviewRating"], second: 5)

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

Figure 6c. Body Code for test_processing_classes.py

```
Ran 2 tests in 0.025s

OK
```

Figure 6d. Test Result for test_processing_classes.py

The test_presentation_classes.py file consists of two tests for the presentation classes module. First, an empty list (employee_data) is set up. The two tests carried out are created as follows:

- test_input_menu_choices – This function simulates the if the user selects “2” (Enter and display data) from the menu of choices. sample data is created and written to the temporary file. The assertEqual function is used to assert that the proper action for selection “2” is returned.
- test_input_employee_data – This function simulates user inputs for employee data being the first name, last name, review date and review rating. The simulation ensures data is not added to the list if an error is made such as the review date not being in ISO 8601 format or the review rating not being an integer.

All of the code as described above for test_processing_classes.py and the test results are shown in **Figure 7a-7c**.

```python
import unittest
from unittest.mock import patch
from presentation_classes import IO
from data_classes import Employee

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', '2024-08-22', 4]):
            IO.input_employee_data(self.employee_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: '2024-08-22')
            self.assertEqual(self.employee_data[0].review_rating,  second: 4)
```

Figure 7a. Body Code for test_presentation_classes.py

```
        # Simulate invalid review date (not in iso format)
        with patch( target: 'builtins.input', side_effect=['Alice', 'Smith', 'invalid date']):
            IO.input_employee_data(self.employee_data, Employee)
            self.assertEqual(len(self.employee_data),  second: 1)  # Data should not be added due to invalid input

        # Simulate invalid review rating (not an int)
        with patch( target: 'builtins.input', side_effect=['Alice', 'Smith', '2024-08-22', 'invalid rating']):
            IO.input_employee_data(self.employee_data, Employee)
            self.assertEqual(len(self.employee_data),  second: 1)  # Data should not be added due to invalid input

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

Figure 7b. Body Code for test_presentation_classes.py

```
That value is not the correct type of data!

-- Technical Error Message --
Review date must be in ISO 8601 format (YYYY-MM-DD).
Inappropriate argument value (of correct type).
<class 'ValueError'>
That value is not the correct type of data!

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


Ran 2 tests in 0.011s

OK
```

Figure 7c. Test Result for test_presentation_classes.py

The test_data_classes.py file consists of seven tests for the data classes module. First, a class is created called TestPerson with the following functions:

- test_person_init which tests the constructor in the Person class by asserting the sample name is as expected.
- test_person_invalid_name which tests the first and last name validations to assert it is as expected.
- test_person_str which tests the __str__() magic method to assert it is as expected.

Next, another class is created called TestEmployee with the following functions:

- test_employee_init which tests the constructor in the Employee class by asserting the sample name, review date and review rating are as expected.
- test_employee_review_date_type which tests the review date validation to assert it is as expected.
- test_employee_review_rating_type which tests the review rating validation to assert it is as expected.
- test_employee_str which tests the __str__() magic method override to assert it is as expected.

All of the code as described above for test_processing_classes.py and the test results are shown in **Figure 8a-8c**.

```
import unittest
import data_classes


class TestPerson(unittest.TestCase):

    def test_person_init(self):  # Tests the constructor
        person = data_classes.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 = data_classes.Person( first_name: "123",  last_name: "Doe")
        with self.assertRaises(ValueError):
            person = data_classes.Person( first_name: "John",  last_name: "123")

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

Figure 8a. Body Code for test_data_classes.py

```python
class TestEmployee(unittest.TestCase):

    def test_employee_init(self):  # Tests the constructor
        employee = data_classes.Employee( first_name: "Alice",  last_name: "Smith",  review_date: "2024-08-22",  review_rating: 4)
        self.assertEqual(employee.first_name,  second: "Alice")
        self.assertEqual(employee.last_name,  second: "Smith")
        self.assertEqual(employee.review_date,  second: "2024-08-22")
        self.assertEqual(employee.review_rating,  second: 4)

    def test_employee_review_date_type(self):  # Test the review_date validation
        with self.assertRaises(ValueError):
            employee = data_classes.Employee( first_name: "Bob",  last_name: "Johnson",  review_date: "invalid date")

    def test_employee_review_rating_type(self):  # Test the review_rating validation
        with self.assertRaises(ValueError):
            employee = data_classes.Employee( first_name: "Bob",  last_name: "Johnson",  review_date: "2024-08-22",  review_rating: "invalid rating")

    def test_employee_str(self):
        employee = data_classes.Employee( first_name: "Eve",  last_name: "Brown",  review_date: "2024-08-22",  review_rating: 4)  # Tests the __str__() magic method
        self.assertEqual(str(employee),  second: "Eve,Brown,2024-08-22,4")

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

Figure 8b. Body Code for test_data_classes.py

```
Ran 7 tests in 0.009s

OK
```

Figure 8c. Test Result for test_data_classes.py

Running the Python Script

To start, when running the script, the user is prompted to select from the program menu. **Figures 9a – 9e** and **Figures 10a – 10c** show what occurs with each selection as described in the "Creating the Python Script" section when running in the PyCharm IDE and command prompt, respectively. This example includes if the user enters a non-alphabetical name, incorrect date format, incorrect review rating data type, and multiple employee registrations. When properly running the script, to prevent an error from occurring when reading the EmployeeRatings.json file, the following starting data was entered: Vic,Vu,2024-08-22,4.

```
---- 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

------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------



---- 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? J4
That value is not the correct type of data!

-- Technical Error Message --
The last name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>

------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------
```

Figure 9a. Running the Python Script from the PyCharm IDE

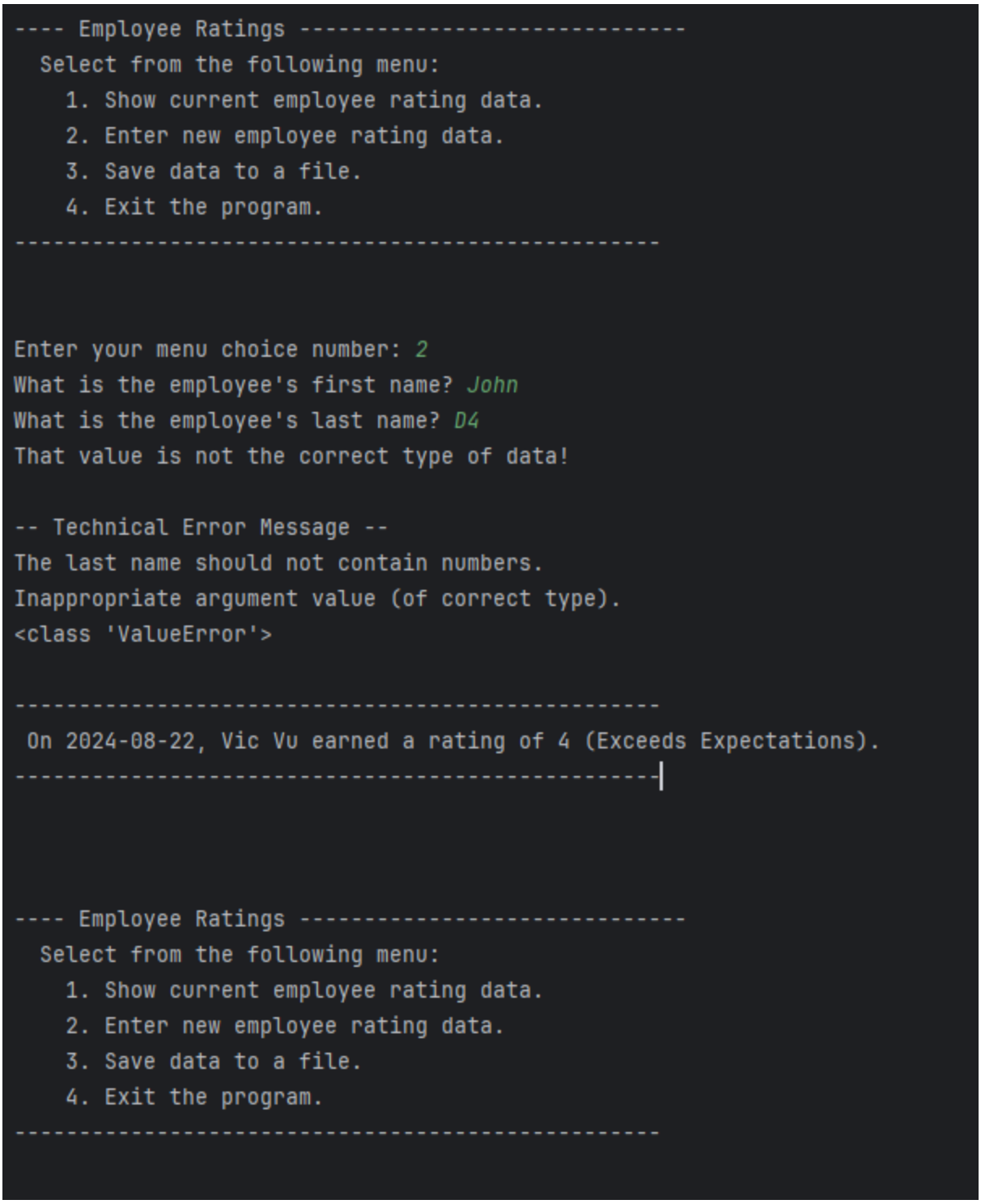

Figure 9b. Running the Python Script from PyCharm IDE

```
Enter your menu choice number: 2
What is the employee's first name? John
What is the employee's last name? Doe
What is the employee's review date? 08-08-2024
That value is not the correct type of data!

-- Technical Error Message --
Review date must be in ISO 8601 format (YYYY-MM-DD).
Inappropriate argument value (of correct type).
<class 'ValueError'>

------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------



---- 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? John
What is the employee's last name? Doe
What is the employee's review date? 2024-08-09
What is the employee's review rating? 5.0
That value is not the correct type of data!

-- Technical Error Message --
invalid literal for int() with base 10: '5.0'
Inappropriate argument value (of correct type).
<class 'ValueError'>

------------------------------------------------
```

Figure 9c. Running the Python Script from PyCharm IDE

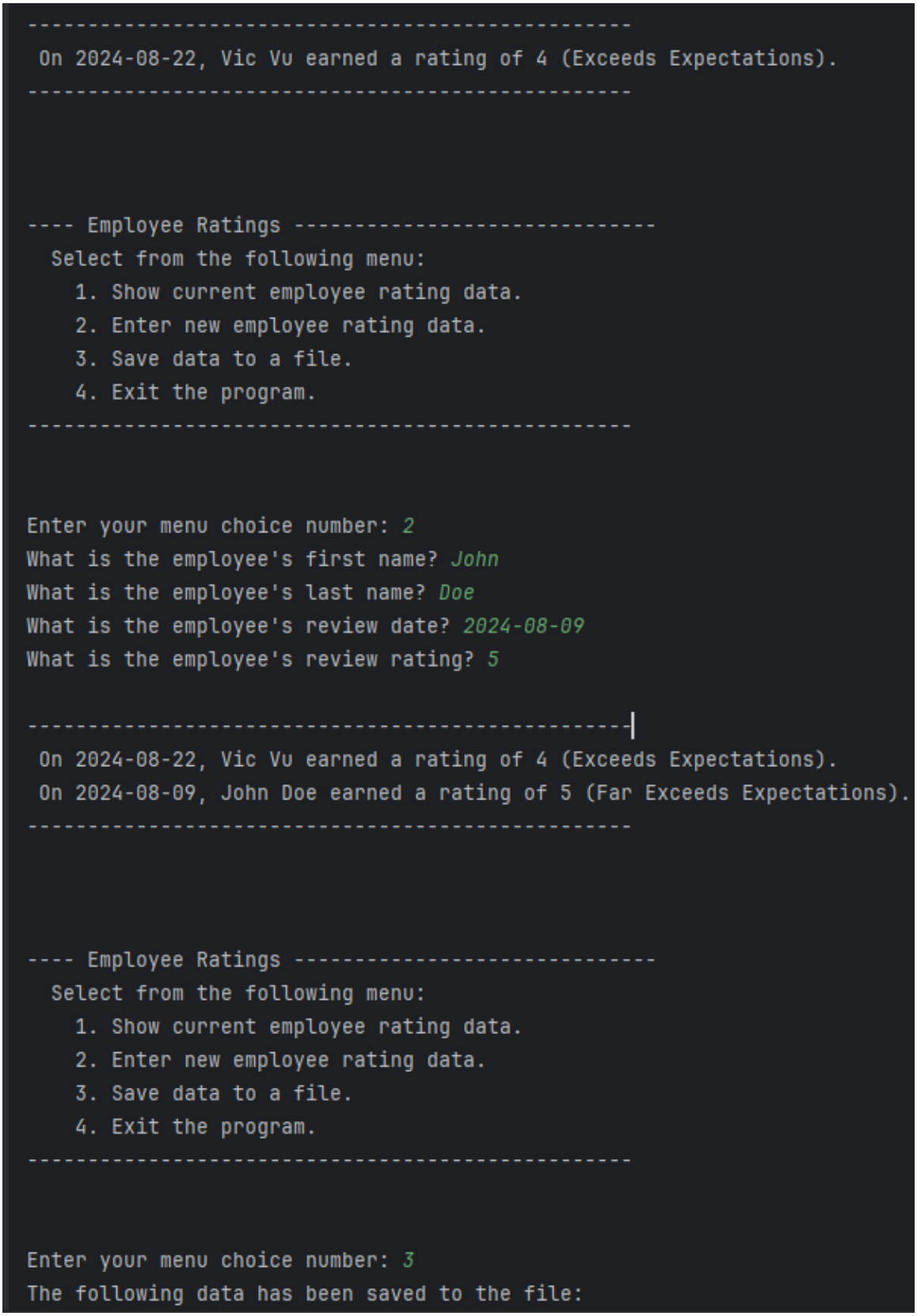

Figure 9d. Running the Python Script from PyCharm IDE

```
---- 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
The following data has been saved to the file:

--------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
 On 2024-08-09, John Doe earned a rating of 5 (Far Exceeds Expectations).
--------------------------------------------------



---- 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
```

Figure 9e. Running the Python Script from PyCharm IDE

```
---- 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

-------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
-------------------------------------------------



---- 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? J4
That value is not the correct type of data!

-- Technical Error Message --
The last name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>

-------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
-------------------------------------------------



---- 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? John
What is the employee's last name? D4
That value is not the correct type of data!

-- Technical Error Message --
The last name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>

-------------------------------------------------
```

Figure 10a. Running the Python Script from the Command Prompt

```
------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------



---- 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? John
What is the employee's last name? Doe
What is the employee's review date? 08-09-2024
That value is not the correct type of data!

-- Technical Error Message --
Review date must be in ISO 8601 format (YYYY-MM-DD).
Inappropriate argument value (of correct type).
<class 'ValueError'>

------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------



---- 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? John
What is the employee's last name? Doe
What is the employee's review date? 2024-08-09
What is the employee's review rating? 5.0
That value is not the correct type of data!

-- Technical Error Message --
invalid literal for int() with base 10: '5.0'
Inappropriate argument value (of correct type).
<class 'ValueError'>

------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
------------------------------------------------
```

Figure 10b. Running the Python Script from the Command Prompt

```
---- 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? John
What is the employee's last name? Doe
What is the employee's review date? 2024-08-09
What is the employee's review rating? 5

-------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
 On 2024-08-09, John Doe earned a rating of 5 (Far Exceeds Expectations).
-------------------------------------------------



---- 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
The following data has been saved to the file:

-------------------------------------------------
 On 2024-08-22, Vic Vu earned a rating of 4 (Exceeds Expectations).
 On 2024-08-09, John Doe earned a rating of 5 (Far Exceeds Expectations).
-------------------------------------------------



---- 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
```

Figure 10c. Running the Python Script from the Command Prompt

As shown below in **Figure 11**, when selecting Option #3, the data is written and saved to the 'EmployeeRatings.json' file.

[{"FirstName": "Vic", "LastName": "Vu", "ReviewDate": "2024-08-22", "ReviewRating": 4}, {"FirstName": "John", "LastName": "Doe", "ReviewDate": "2024-08-09", "ReviewRating": 5}]

Figure 11. Data Stored in .JSON File After Running the Python Script

Summary

The goal of this assignment was to demonstrate the creation of a Python application that prompts the user to select from a program menu in order to allow for entering and displaying multiple employee review ratings along with the ability to read data from/save data to a JSON file using multiple modules, unit testing, and object-oriented programming. Constants, variables, classes, various functions, string formatting, while and for loops, programming menus, conditional logic, classes, objects, constructors, properties, inheritance, exception handling, JSON files, multiple modules and unit testing are utilized in this script in order to achieve the desired outcome.