Alan Martin
02/25/2024
Foundations of Python
Assignment_07
AMart253/IntroToProg-Python-Mod-07 (github.com)

Classes; Getters and Setters with Structured Error Handling

Introduction

In this document I'll be walking though creating python script that uses a loop function to cycle through selections on our menu until we decide to quit our program, reaching the break statement. We'll first use selection #1 to take user input, selection #2 will print the input receive from user, selection #3 will open-save-close file with user data, selection #4 will break the loop and exit the program. We'll create a doc string to describe each class and function(method) to better explain its usage, purpose, and behavior. We'll continue to use JSON files to read data into our program and write data from our program into our JSON file. We'll incorporate structured data handling within those methods.

This week expands on last week's module of classes and functions(methods), and the benefits they serve in programming (encapsulation, modularity, reusability, etc.). We'll create objects in our classes and use the @Property decorator (getter), and the .setter (setter). We'll explore the use of __init__ and __str__ in our class objects. Lastly, we learned about private (double underscore before object name)

Setting python script header

Setting the script header from the start helps document your work. Included I have my title, a description, python version, change log describing who, what, when. I added Python version so I can become more aware of how a new version may affect my program.

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Python 3.12.1
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Alan Martin, 02/24/2024, Created Script
#   Alan Martin, 02/25/2024, Testing and tidying up code
# ------------------------------------------------------------------------------------------ #
```

Import libraries

Since we'll be dealing with Json files, we needed to **import** the **Json** library.

```
import json

```

Defining the Constants

The constants in this program were predefined. The **MENU** constant will become the visual instruction for the user. The **FILE_NAME** constant is set to **Enrollments.json** and will end up saved in the same folder as my program.

```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments.json"

```

Defining the Variables

These variables names have been predefined, but their values have not been set. **menu_choice** holds the choice made by the user and directs the loop. **Students variable** holds a **table of student data**

```
25  # Define the Data Variables
26  menu_choice: str = ""  # Hold the choice made by the user.
27  students: list = []  # a table of student data
28
29
```

Identifying our classes and Methods

Our first class is **FileProcessor.** The triple double-quotes (” ” ”) signals the docstring, which is where you want to define the classes’ purpose, usage, and behavior, along with your change log.

```
31  class Person:
32      """
33      A person class to hold student data and interacts with JSON file
34
35      ChangeLog: (Who, When, What)
36      RRoot,1/1/2030,Created Script
37      Alan Martin, 02/24/2024 edited first_name, last_name, course_name attributes
38      """
39
```

Add properties to constructor by (__init__)

__init__ is a constructor method in python. It is called when a new instance of the class is initialized, giving the instance the necessary attributes, and helps set up the actions for the object to function properly.

```
40
41  # TODO Add first_name and last_name properties to the constructor (Done)
42      def __init__(self, first_name: str = "", last_name: str = ""):  # Move the fields into constructors (now attributes)
43          self.first_name = first_name  # Set the attribute using the property for validation
44          self.last_name = last_name  # Set the attribute using the property for validation
45
```

Use getter and setter methods (decorators)

To access the getter method, you use the **@Property** decorator. The getter method allows you to ***access*** the **first_name attribute value,** where the **.setter method** allows you to *modify* the **first_name method attribute values.** From line 46 to 58, you can also see some error handling, validation, and return statement. On line 50 you can see the **self.__first_name.** That double underscore before the **firstname** is an indication that the variable or method is private in the class. In python it essentially makes it harder to be accessed from outside the class.

```
46  # TODO Create a getter and setter for the first_name property (Done)
47
48      @property
49      def first_name(self):
50          return self.__first_name.title()  # __ for private, .title() for title casing
51
52      @first_name.setter
53      def first_name(self, value: str):
54          if value.isalpha() or value == "":  # Alphanumeric validation code
55              self.__first_name = value
56          else:
57              raise ValueError("The first name should not contain numbers.")
58
```

Use getter and setter methods (decorators)

*Read previous statement that covers the same exact information.

```python
# TODO Create a getter and setter for the last_name property (Done)

    @property
    def last_name(self):
        return self.__last_name.title()  # __ for private, .title() for title casing

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":  # Alphanumeric validation code
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")
```

__str__ method to return data

The **__str__ method** is a special method used to return a string representation of an object. When the **__str__ method** converts the object data, it becomes much easier for the user to read.

```python
# TODO Override the __str__() method to return Person data (Done)
    def __str__(self):
        return f"{self.first_name},{self.last_name}"

```

Created Class Student and doc string

Just as done before, a new Class is created and doc string to define its purpose, use, and behavior. This **Student class** inherits **Person class objects.** This can be seen in the “class Student(Person)”

```python
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person):
    """
     A student class to hold student data and interacts with JSON file.

    ChangeLog: (Who, When, What)
    RRoot,1/1/2030,Created Script
    Alan Martin, 02/24/2024, added methods to include course_name to Class student
    """

```

Class initializer __init__ method

Same **__init__** method as we discussed above, the **super().__init__ method** is a way to call the constructors from the super class (in this case it's Person Class). This method is used to inherit attributes and behaviors from the super class, in addition to additional attributes and behaviors defined in the student class.

```
87    # TODO call to the Person constructor and pass it the first_name and last_name data (Done)
88        def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
89            super().__init__(first_name=first_name, last_name=last_name)
90
```

Add course_name property to the Student Class

Self.course_name = course_name

```
91
92    # TODO add a assignment to the course_name property using the course_name parameter (Done)
93            self.course_name = course_name
94
```

Use getter and setter methods

As we did in the above **Person Class**, use the **@Property** decorator as the getter method to access the attribute values. Use the **.setter method** to manipulate the attribute values**, __** to make **course_name** private and harder to access from outside the method. *.**title() makes the return value in tatle case**

```
95    # TODO add the getter for course_name (Done)
          3 usages (2 dynamic)
96        @property
97        def course_name(self):
98            return self.__course_name.title()
99
00    # TODO add the setter for course_name (Done)
          3 usages (2 dynamic)
01        @course_name.setter
02        def course_name(self, value: str):
03            self.__course_name = value
04
```

Define Class Student __str__ method

Makes the data easier for the user to read when it's returned to them. **Student class** includes the **{self.course_name}**

```
106     # TODO Override the __str__() method to return the Student data (Done)
107     def __str__(self):
108         return f"{self.first_name}, {self.last_name}, {self.course_name}"
109
110
```

Created Class FileProcessing w/ docstring

```
111     # Processing ------------------------------------- #
112     class FileProcessor:
113         """
114         A collection of processing layer functions that work with Json files
115
116         ChangeLog: (Who, When, What)
117         RRoot,1.1.2030,Created Class
118         """
            1 usage
```

@staticmethod, class FileProcessor, Method read_data_from_file

Method is the term used when a function sits inside a class, otherwise we'll call it a function. In this foundations class it may be used interchangeably, but let it be known that there is a difference. @staticmethod means that the method can be called from the class without an instance of the class. In this method, we're opening our json file while using some error handling seen in the try except, finally functions. b **IO.output_error_messages** in the except function will call the method. Inside the parenthesis on the method are parameters (**file_name**: str, **student_data**: list), arguments for these parameters will be seen later when the program runs.

```
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ 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

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        file = ""
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)

        finally:
            if not file.closed:
                file.close()
        return student_data

```

@staticmethod, class FileProcessor, Method write_data_to_file

Reference @staticmethod, class FileProcessor, Method read_data_from_file. This class method writes student_data table to file using json.dump. This file should be saved in the same folder as your program. Error handling is used here as well, along with calls to other methods.

```
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        Alan Martin, 02/24/2024, added empty file variable
        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """
        file = ""

        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            IO.output_student_and_course_names(student_data=student_data)
        except Exception as e:
            message = "Error: There was a problem with writing to the file.\n"
            message += "Please check that the file is not open by another program."
            IO.output_error_messages(message=message, error=e)
        finally:
            if file.closed is False:
                file.close()
```

Identifying our classes and Methods. Class IO

This class manages user input and output.

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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    RRoot,1.2.2030,Added menu output and input functions
    RRoot,1.3.2030,Added a function to display the data
    RRoot,1.4.2030,Added a function to display custom error messages
    """

    @staticmethod
```

@staticmethod, class IO, Method output_error_messages

This class method helps with error handling messages. In class **FileProcessor**, you'll see both methods **read_data_from_file** and **write_data_to_file** have calls to **IO.output_error_messages.** The **output_error_messages** method is called when the **except** function is triggered in the **FileProcessor** class methods, printing messages specific to the error.

```
184     @staticmethod
185     def output_error_messages(message: str, error: Exception = None):
186         """ This function displays a custom error messages to the user
187
188         ChangeLog: (Who, When, What)
189         RRoot,1.3.2030,Created function
190
191         :param message: string with message data to display
192         :param error: Exception object with technical message to display
193
194         :return: None
195         """
196         print(message, end="\n\n")
197         if error is not None:
198             print("-- Technical Error Message -- ")
199             print(error, error.__doc__, type(error), sep='\n')
```

@staticmethod, class IO, Method output_menu

This method displays our MENU constant for the user.

```
        1 usage
201     @staticmethod
202     def output_menu(menu: str):
203         """ This function displays the menu of choices to the user
204
205         ChangeLog: (Who, When, What)
206         RRoot,1.1.2030,Created function
207
208
209         :return: None
210         """
211         print()  # Adding extra space to make it look nicer.
212         print(menu)
213         print()  # Adding extra space to make it look nicer.
214
        1 usage
```

@staticmethod, class IO, Method input_menu_choice

This method takes the user input and returns it to the user. If the user input isn't string(1, 2 ,3, 4) **raise Exception** print a reminder to the user that only 1, 2, 3, 4 are accepted. If any other errors occur, **IO.output_error_messages** method is called. Again, the user choice is **returned**

```
215     @staticmethod
216     def input_menu_choice():
217         """ This function gets a menu choice from the user
218
219         ChangeLog: (Who, When, What)
220         RRoot,1.1.2030,Created function
221
222         :return: string with the users choice
223         """
224         choice = "0"
225         try:
226             choice = input("Enter your menu choice number: ")
227             if choice not in ("1", "2", "3", "4"):  # Note these are strings
228                 raise Exception("Please, choose only 1, 2, 3, or 4")
229         except Exception as e:
230             IO.output_error_messages(e.__str__())  # Not passing e to avoid the technical message
231
232         return choice
233
```

@staticmethod, class IO, Method output_student_and_course_names

This function displays all student data including their first name, last name, and course. It uses a **for loop, r**eading and printing each student for the user to see.

```
        2 usages
234     @staticmethod
235     def output_student_and_course_names(student_data: list):
236         """ This function displays the student and course names to the user
237
238         ChangeLog: (Who, When, What)
239         RRoot,1.1.2030,Created function
240
241         :param student_data: list of dictionary rows to be displayed
242
243         :return: None
244         """
245
246         print("-" * 50)
247         for student in student_data:
248             print(f'Student {student["FirstName"]} '
249                   f'{student["LastName"]} is enrolled in {student["CourseName"]}')
250         print("-" * 50)
251
```

@staticmethod, class IO, Method input_student_data

This method takes input for a student's first name, last name, and course. Error handling can be seen in **if not student_first_name.isalpha()** and **if not student_last_name.isalpha().** This line of code is flagged if the user enters anything other than alphabetic letters. After first name, last name, and course have been input by the user, that data is stored into the **student: dict,** and then appended to the **student_data: table.** Specific error handling messages can be seen on line 141 and 143. **Return student_data**

```
253     @staticmethod
254     def input_student_data(student_data: list):
255         """ This function gets the student's first name and last name, with a course name from the user
256
257         ChangeLog: (Who, When, What)
258         RRoot,1.1.2030,Created function
259
260         :param student_data: list of dictionary rows to be filled with input data
261
262         :return: list
263         """
264
265         try:
266             student_first_name = input("Enter the student's first name: ")
267             if not student_first_name.isalpha():
268                 raise ValueError("The last name should not contain numbers.")
269             student_last_name = input("Enter the student's last name: ")
270             if not student_last_name.isalpha():
271                 raise ValueError("The last name should not contain numbers.")
272             course_name = input("Please enter the name of the course: ")
273             student = {"FirstName": student_first_name,
274                        "LastName": student_last_name,
275                        "CourseName": course_name}
276             student_data.append(student)
277             print()
278             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
279         except ValueError as e:
280             IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
281         except Exception as e:
282             IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
283         return student_data
```

Read the JSON file data into a list of lists

Students variable now holds the JSON file data from the **read_data_into_file** method, using arguments **FILE_NAME, students.**

```
285 # Start of main body
286
287 # When the program starts, read the file data into a list of lists (table)
288 # Extract the data from the file
289 students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
290
```

While loop and present MENU

The **while loop** starts and **IO.output_menu** presents the **MENU** constant shown the argument. Whichever number the user selects will direct them in the **MENU** and will also be stored in **menu_choice.**

```
291     # Present and Process the data
292     while True:
293
294         # Present the menu of choices
295         IO.output_menu(menu=MENU)
296
297         menu_choice = IO.input_menu_choice()
298
```

If conditional statement and menu_choice #1

If menu_choice == 1, **students** variable stores data collected by the **IO.input_student_data method. Continue** starts the **while loop** over

```
300     if menu_choice == "1":  # This will not work if it is an integer!
301         students = IO.input_student_data(student_data=students)
302         continue
303
```

elif conditional statement and menu_choice #2

elif menu_choice == 2, output **students** data to the user. **Students** is the argument for the **student_data** parameter.

```
304         # Present the current data
305         elif menu_choice == "2":
306             IO.output_student_and_course_names(students)
307             continue
308
```

elif conditional statement and menu_choice #3

elif menu_choice == 3, save data to JSON file. The **FileProcessor.write_data_to_file** call uses **FILE_NAME** and **students** arguments. In the method, it takes the data stored in **students** variable and stores it into the **FILE_NAME** constant**.**

```
309     # Save the data to a file
310     elif menu_choice == "3":
311         FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
312         continue
313
```

elif conditional statement and menu_choice #4

elif menu_choice == 4, break out of the loop.

```
314         # Stop the loop
315         elif menu_choice == "4":
316             break  # out of the loop
317         else:
318             print("Please only choose option 1, 2, or 3")
319
```

Print to show program has ended

This print statement let the user know that the program has ended.

```
319
320     print("Program Ended")
321
```

Summery

In this assignment our constants were predefined, and our variable names were given but not assigned. We read a json file into a list in our program, allowing us to build onto an existing file, we also read stored data back into our json file to be stored. Option 1 allowed use to receive user input and collect data that we used to append our list. Option 2 read the students list back to us, ensuring that data was collected. Option 3 allowed us to save data that was collected and option 4 allowed us to save and close the program.

We learned about getter and setter methods and how they allow use to access attribute values and manipulate them. Wh learn about the __init__ initializer method and the __str__ method. The more I've learned the more I've been able to clean up my code which makes it easier to read, more modular, better for testing, and is more efficient .

References

Common Header Format in Python | Delft Stack
Reading and Writing to text files in Python - GeeksforGeeks
Python if-else Statement (tutorialspoint.com)
Writing Professional Papers (youtube.com)
What Is JSON | Explained (external site)
Exceptions in Python - Python Tutorial (external site)
GitHub Tutorial - Beginner's Training Guide (external site)
Python Exceptions: An Introduction (external site)
Functions - Learn Python - Free Interactive Python Tutorial
Let's Learn Python - Basics #6 of 8 - Functions - YouTube
Python OOP Tutorial 1: Classes and Instances (external site)
PyCharm Version Control w/Git and GitHub (external site)
3 Simple Ways ChatGPT Can Make You a Better Coder (external site)
python classes and objects