Natalie Ferri
August 013, 2024
Foundations Of Programming: Python
Assignment 07
https://github.com/nferri/IntroToProg-Python-Mod07-

Creating Functions for the Python Program

Introduction

This document provides insights into how I created a Python-based code to process and save data to a JSON file using user input, lists, and a dictionary grouped by classes and functions. The program leverages data classes and implements structured error handling to ensure clarity while managing student data.

Writing A Registration Program

The course builds upon the previous week's code to introduce new techniques. This week, we continued writing to a JSON file using classes and functions. The assignment is to create a Python program that uses classes and functions for the input, open, write/dump, close, lists, dictionaries, print, and error handling functions to capture data from a user and display messages to a user about course registration. The documentation will be broken into three sections: assigning variables, executables, and testing.

Assigning Variables

Before defining constants, we must import the JSON module to work with the JSON file. Using the outline from the assignment deliverables, I assigned the constant variables for FILE_NAME and MENU. FILE_NAME defines the name of the JSON to be written to by the program from user input. MENU is an open string and will have text written for the output within the logic statement.

```
import json

# Define the Data Constants
FILE_NAME: str = "Enrollments.json"

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.

----------------------------------------

'''
```

Instead of using predefined variables in the header of the script, we introduced global variables in the header script and local variables within the defined functions. This week, we still use the menu_choice and students variables. Menu_choice, is variable used to store the user's choice from the menu. Students is alist that holds instances of the Student class, representing the enrolled students.

```
# Define the Data Variable
menu_choice: str = "" # choice made by the user.
students: list = [] # a table of student data
```

Executable

This week we finished creating a registration script with using classes and functions to execute the main program. We created two new classes: Person and Student. We introduced the __init__ and __self__ functions today which initialize objects. See an example of the __init__ initialization in Figure 1.

```
# Define the Person Class
class Person:
    """
    A class representing person data with error identification.

    ChangeLog:
    Natalie Ferri, 8/12/2024, Created class Person

    proterty first_name: string data
    property last_name: string data

    Inherited by:
        Student class
    """

    def __init__(self, first_name: str = '', last_name: str = ''):
        """
        Initialize a Person object with first and last names.

        :param first_name: The first name of the person
        :param last_name: The last name of the person
        """
        self._first_name = ''
        self._last_name = ''
        self.first_name = first_name
        self.last_name = last_name
```

Figure 1:__init__

Class Person represents an individual with personal attributes and includes structured error handling for data validation. The variables first_name and last name are the input names and validation checks that the names are only alphabetic characters.

Class Student is dependent on class Person and inherits the student's first and last name to include the inputted course name, Figure 2.

```python
# Define the Student Class
class Student(Person):
    """
    A class representing student data with error identification.

    ChangeLog:
    Natalie Ferri, 8/12/2024, Created class Student

    property course_name: string data
    """

    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        """
        Initialize a Student object with first and last names and course name.

        :param first_name: The first name of the student
        :param last_name: The last name of the student
        :param course_name: The name of the course the student is enrolled in
        """
        super().__init__(first_name, last_name)
        self.course_name = course_name

    @property
    def course_name(self) -> str:
        """
        Get the course name, capitalized.

        :return: The course name in title case
        """
        return self._course_name.title()

    @course_name.setter
    def course_name(self, value: str) -> None:
        """
        Set the course name with validation.

        :param value: The course name to set
        :raises ValueError: If the value contains invalid characters
        """
        if all(char.isalnum() or char.isspace() for char in value) or not value:
            self._course_name = value
        else:
            raise ValueError("Course name must be alphanumeric")
```

Figure 2: class Student

Class FileProcessor continues to manage the file operations for the student data. These include reading and writing to the Enrollments.JSON file. Class IO continues to handle error messaging if encountered through execution, Figure 3.

```python
# Define the IO Class
class IO:
    '''
    This class includes functions that display a menu with choices, accepts user input,
        present user input, and has error handling.
    '''

    @staticmethod
    def output_error_messages(message: str, error: Exception = None) -> None:
        """
        Output custom error messages to the user.

        :param message: The custom message to display
        :param error: Optional exception to display technical details
        """
        print(f"{message}\n")
        if error:
            print("-- Technical Error Message --")
            print(f"{error}\n{error.__doc__}\n{type(error)}")

    @staticmethod
    def output_menu(menu: str) -> None:
        """
        Display the menu of choices to the user.

        :param menu: The menu string to display
        """
        print(f"\n{menu}\n")

    @staticmethod
    def input_menu_choice() -> str:
        """
        Get and validate the user's menu choice.

        :return: The valid menu choice entered by the user
        """
```

Figure 3: class IO snippet

Lastly, we end the script with the main program logic that calls each class to run when the menu_choice is selected, Figure 4.

```python
# Main Program
students = FileProcessor.read_data_from_file(FILE_NAME, students)  # Load existing student

while True:
    IO.output_menu(MENU)  # Display the menu
    menu_choice = IO.input_menu_choice()  # Get the user's choice

    if menu_choice == "1":
        students = IO.input_student_data(students)  # Register a new student
    elif menu_choice == "2":
        IO.output_student_and_course_names(students)  # Show current data
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(FILE_NAME, students)  # Save data to a file
    elif menu_choice == "4":
        break  # Exit the loop
    else:
        print("Please choose a valid option.")

print("Program Ended")
```

Figure 4: Main program

Testing

I struggled through Assignment07.py because I originally moved my global variables after class IO, which caused type errors with my functions for executing menu_choice 2. I was excited that

menu_choice 1 had executed after creating class Person and Student, but my happiness was brief! I spent a while reviewing the assignment 07 answer video and was unable to resolve the issue. I returned to Assignement06.py and wrote in the class Person and Student, and then edited the dependant functions in Class FileProcessor and IO. Because I did not move the global variables this time, I had better luck with the script running. Functions are very hard for me to understand what needs to go where in order to execute correctly.

Although I struggled with Assignment 07, I did not struggle as much as I did with Assignement 06. Luckily for my work, I will not have to write classes and functions, since our work does not need a lot of iteration performing functions on multiple objects. Instead, I do specific actions to singular datasets. It is still helpful to know how class and defined functions work.

Lastly, after getting my script to execute properly, I used chatgpt as suggested in the assignment videos, to make my script more pythonic. Pythonic is a buzz word my boss loves to use and I wanted to test it out. It worked well for the code and didn't make too many changes. The major change I noticed was using extended logic:

```
# Convert each dictionary to a Student object and add to the list
  student_data.extend(Student(
      first_name=student["FirstName"],
      last_name=student["LastName"],
      course_name=student["CourseName"]
   ) for student in list_of_dict_data)
```

We have not covered this, but I can see that instead of breaking the logic into singular code lines outside of the parentheses, it creates one long argument contained within multiple parentheses. I think chatgpt is a useful resource for improving upon code and I enjoyed testing it.

Summary

In conclusion, I detailed my process for writing a Python-based program that uses defined functions within classes to allow the user to interact with the program and how to incorporate lists to store and call data. The script uses the user's input to save data to a JSON file and presents it to them in print statements. Multiple users can submit information to be saved since the loop allows the variables to be reassigned as the user inputs names. The JSON uses the dump function to write each new input saved. Finally, the script concludes when the user selects the break function for the code.

```
exit()  # end writeup
```