Sonali R. Mishra

May 25 2025

Foundations of Programming: Python

Assignment for Module 6

https://github.com/codebeaker/IntroToProg-Python-Mod06/

Assignment 6: Functions, Classes, and Modular Code

Introduction

Up to this point, we have been writing ever more complex scripts. Module 6 teaches us a more modular way of writing code: functions. We also learn classes as a way of organizing functions into meaningful collections and separating types of concerns (e.g., reading and writing data vs. manipulating data within the program). In this assignment, we reorganize the script we have been iterating on for enrolling students via a menu into functions grouped into classes.

Functions

Functions are small modular collections of actions you want the computer to take on your data, for instance reading it into memory from a file, writing it to a file, or performing some specific set of operations. The goal is to write functions to be limited in scope, to make the resulting code more modular, rather than to write large complex all-encompassing functions that are hard to parse. Notably, because variables are defined locally within functions (and then passed outside the function via a return statement), it no longer makes sense to declare all variables at the start of the program, simplifying the code.

Because I have continued in this assignment to have an extra feature in my program, namely the ability for users to see both saved and unsaved data, I wrote the output_student_courses() function with two parameters rather than one: the list of dictionaries that contains the data, and a message that displays above the data that can inform the user what they are looking at. This structure allows me to flexibly use the same function to display saved or unsaved data, and to pass in a contextualized message as a second argument to make clear to the user what they are looking at and remind them to save unsaved data.

Classes

Classes are collections of functions. In this assignment we use classes to organize our functions; however, we are not creating objects, and are in fact using the @staticmethod decorator to avoid having to create objects. However, classes are also useful for creating objects. For instance, we could in theory use classes to create synthetic students with randomly assigned GPAs and course names to populate fake data into our initial json file. For this assignment, however, it is much easier to manually populate the data.

The code

Below is the code for this assignment.

```
# ----------------------------------------------------------------------------
--------------- #
# Title: Assignment06
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Sonu Mishra,5/25/25, Edited Script
# ----------------------------------------------------------------------------
--------------- #
import json
import io as _io

# 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.
-----------------------------------------
'''

#--------- Define the Data Variables
menu_choice: str  # Hold the choice made by the user.
students: list = []  # a table of student data that has been saved
students_unsaved: list = [] #to hold data that has not been saved yet

#--------- Define the classes and methods

#------PROCESSING LAYER------
class FileProcessor:
    """
    Functions to read and write to and from a JSON file.
    ChangeLog: (Who, When, What)
    Sonu Mishra, 5/25/25, created class
    """
    @staticmethod
    def read_data_from_file(file_name: str, students: list):
        '''
        This function reads data from a JSON file and stores it.
        :param file_name: str
        :param students: list
        :return:students: list
        '''
        try:
            file = open(file_name, "r")
            students = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages("Text file must exist before running
this script!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return students

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        '''
        This function writes data to a JSON file.
        :param file_name: string
        :param student_data: list
        :return: None
        '''
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages("Please check that the data is a valid
JSON format", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()

#Presentation -----------------------------------#
class IO:
    """
    Functions to manage input and output including menu display, menu choice,
    error message displays, storing data in memory, and storing data
    in saved or unsaved lists.
    ChangeLog: (Who, When, What)
    Sonu Mishra, 5/24/25, created script
    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays the custom error messages to the user
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message --")
            print(error, error.__doc__, type(error), sep="\n")

    @staticmethod
    def output_menu(menu: str):
        """
        defining the menu output function
        :param menu: str
        :return: None
        """
        print()  # Adding extra space to make it look nicer.
        print(menu)


    @staticmethod
    def input_menu_choice():
        """
        This function gets a menu choice from the user
        :return: string with the user's choice
        """
        user_choice = "0"
        try:
            user_choice = input("Enter your menu choice number:").strip() #in
case they put in whitespace
            if user_choice not in ("1","2","3","4"):
                raise Exception("Try again! Please, choose only 1, 2, 3, or
4")
        except Exception as e:
            IO.output_error_messages(e.__str__()) #Not passing e to avoid the
technical message
        return user_choice

    @staticmethod
    def input_student_data(student_data: list):
        '''
        This function collects the user's data and stores it in a list
        '''
        try:
            # Input the data
            student_first_name = input("What is the student's first name? ")
            #allow letters and spaces but not numbers
            if not student_first_name.replace(" ","").isalpha():
                raise ValueError("The first name should contain only letters
and numbers.")

            student_last_name = input("What is the student's last name? ")
            #allow letters and spaces but not numbers
            if not student_last_name.replace(" ","").isalpha():
                raise ValueError("The last name should contain only letters
and numbers.")

            course_name = input("What is the course name? ")
            student = {"FirstName": student_first_name,
                       "LastName": student_last_name,
                       "CourseName": course_name}
            student_data.append(student)
        except ValueError as e:
            IO.output_error_messages("That value is not the correct type of
data!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        return student_data


    @staticmethod
    def output_student_courses(student_data: list, message: str):
        """
        This function displays data to the user
        with a message to contextualize it
        ChangeLog: (Who, When, What)
        RRoot,1.3.2030,Created function
        RRoot,1.4.2030,Added code to toggle technical message off if no
exception object is passed
        me, 5/24/25, edited code
        :return: None
        """
        print()
        print(message)
        for student in student_data:
            print(f"Student {student['FirstName']} {student['LastName']} is "
                  f"enrolled in {student['CourseName']}.")
        print()



    @staticmethod
    def move_unsaved_to_saved(unsaved_data: list, saved_data: list):
        '''
        This function appends items from the 'unsaved' data list to the
        'saved' data list, and clears the 'unsaved' data.
        :param unsaved_data:
        :param saved_data:
        :return: saved_data: list
        '''
        for item in unsaved_data:
            saved_data.append(item)
        unsaved_data.clear()
        return saved_data



# Beginning of the main body of this script

#Read in data from file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
students=students)


# Present and Process the data
while (True):

    # Present the menu of choices
    print(MENU)
    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1":
        students_unsaved = IO.input_student_data(students_unsaved)
        #show the user what they have entered but not yet saved
        IO.output_student_courses(students_unsaved, "Don't forget to save
this data:\n")
```

```
    # Present the current data
    elif menu_choice == "2":

        #show the already saved enrollments
        IO.output_student_courses(students, "This is the saved data:\n")

        #Show the unsaved data from the current session
        if students_unsaved != []:
            IO.output_student_courses(students_unsaved, "This data has not 
yet been saved:\n")
        continue

    # Save the data to a file
    elif menu_choice == "3":

        # Add unsaved data to saved list and clear unsaved list
        IO.move_unsaved_to_saved(students_unsaved, students)

        #write saved list data to file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, 
student_data=students)

        # Display what has been saved
        IO.output_student_courses(students, "This data has been saved:\n")


    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop

print("Program Ended")
```

Figure 1: Code for Assignment 6.

Summary

In this module we learned about functions and classes. This assignment refactors the program we had previously written by moving tasks to functions and organizing the functions in classes. The goal is to separate types of concerns and write clean, readable, modular code.