Lisa Robinson

November 16, 2023

Foundations of Programming: Python

Assignment 06

https://github.com/Lisa-jr-batista/IntroToProg-Python-Mod06

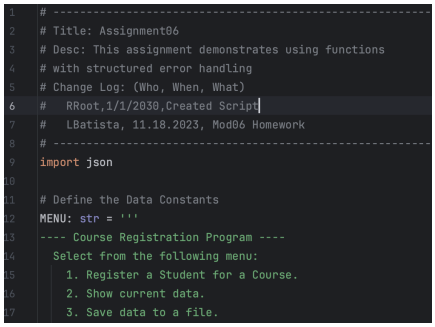
Functions

Introduction

The main focus this week was introducing and practicing three common techniques for improving scripts. Those techniques are functions, classes and the separation of concerns programming pattern. The first we covered were functions - the reason why you might use a function in your code is because it allows you to break down the code into smaller steps (this would be essential when tackling a major program) and it allows you to reuse the function over and over again within the code. A frustration I've had during this journey is how detailed it feels like I need to be for every single step and that I occasionally have to repeat commands– using a function might relieve some of that frustration. Next we worked through parameters and arguments (which I learned are not the same thing and I should correct people who use the incorrect term). A parameter acts as a local variable within the function and might be preferable to use than global variable because they are more flexible and reusable because it is more maintainable. A parameter becomes an argument when there is an operator added that assigns value. We then moved into classes, and this is where things started to get more shaky for me. Up until this point it felt like each new term and idea built on the last, and while classes are helpful if you want to group functions, the coding does not feel as intuitive as earlier concepts. Finally we worked with separation of concerns and making sure our code stays organized.

Writing the Program

I started this week with the Assignment 6 starter python file. After last week's mistake of not using JSON, I double checked that it was imported. Figure 1 shows my header and imports.



```
# ------------------------------------------------------------
# Title: Assignment06
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   LBatista, 11.18.2023, Mod06 Homework
# ------------------------------------------------------------
import json

# 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.
```

Figure 1 - Script header

I decided to give the starter code a run to see if I needed to troubleshoot any issues and right away got a "file not found" error. The "enrollments.json" didn't exist yet despite having used it for other modules. I believe this was an error holdover from last week as I accidentally used CSV. Rather than immediately create the file to use, I wanted to see if I could add in the correct command with the error messages to have my program write the json file if it didn't already exist. I could not remember exactly how to accomplish this, so I went to our class's shared GitHub and remembered that Sabrina had created a json file in her demo. I took a look at her GitHub repository and found the code I needed (Thanks, Sabrina!). Figure 2 shows the output error messages function in the class IO, and figure 3 shows my added code to create the file if it did not already exist with the read data from file function in class FileProcessor.

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

    Changelog: (who, when, what)
    LBatista, 11.18.2023, created the function

    :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 2 - output error messages function in class IO

```
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """
    This function reads the file and raises an error if the json file doesn't exist.
    :param file_name: Enrollments.json
    :param student_data: student first name, student last name, course
    :return: student data
    """
    try:
        file = open(file_name, "r")
        student_data = json.load(file)
        file.close()
    except FileNotFoundError as e:
        IO.output_error_messages("Text file must exist before running this script!\n Creating it...")
        with open(FILE_NAME, "w") as file:
            json.dump(students, file)
    except Exception as e:
        IO.output_error_messages("There was a non-specific error!\n")
    finally:
        if file.closed == False:
            file.close()
    return student_data
```

Figure 3 - Added file creation (if it didn't exist) in the read data from file function in the file processor class

Next I made sure the menu options would display for the user and added the output menu function in the IO class. I adjusted the code that calls for the menu to connect to the correct function and then tested my code to ensure that things were working up until now. I had to make a few adjustments due to syntax errors, but after some debugging, I was able to get the menu to consistently work and I was able to run through the rest of the starter code. Onto the menu choices! Starting with option 1, I moved the code to enter in student data to the IO class and created a new function. I also adjusted the error handling so it

would connect to the output error messages function from earlier. Then I updated the call code to connect to this function. Figure 4 shows the "input_student_data" function.

```
@staticmethod
def input_student_data(student_data:list):
    """ This function allows the user to input data for new students

    Changelog: (who, when, what
    LBatista, 11.19.2023, created function
    :return: str
    """
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
        course_name = input("Please enter the name of the course: ")
        student_data = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        students.append(student_data)
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    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)
```

Figure 4 - new input student data function for when option 1 is selected with adjusted error handling.

Next I moved into menu choice 2 for showing the user the current data in the file. Originally I played around with calling the first function "read_data_from_file" in the FileProcessor class, but when I ran the code, nothing was displayed. I realized that I needed an f-string in the function in order to format the information and return it to the user. Since there was already a good amount of code in the "read_data_from_file" function, I decided that rather than continue to mess around with it, it would make more sense to create a new function in the IO class. I titled it "output_student_courses" and copy and pasted the formatting from down below into the function and adjusted the option 2 call code. Figure 5 shows the new function. I ran the code from here to test if things still worked.

```
145     @staticmethod
146     def output_student_courses(student_data:list):
147         """ This function displays the current data in the file to the user
148
149         Changelog (who, when, what)
150         LBatista, 11.19.2023, created function
151
152         return: None
153         """
154         print("-" * 50)
155         for student in students:
156             print(f'Student {student["FirstName"]} '
157                   f'{student["LastName"]} is enrolled in {student["CourseName"]}')
158         print("-" * 50)
```

Figure 5 - Output student courses function

After this, I was nearly done with my code, but when I ran it, I discovered an issue with outputting the information. When I ran the code and selected "2" for "show current data," the code was not correctly reading my file and just printed the dashes I asked for. I discovered there were some errors that needed to be fixed in order to get the code working correctly. First after looking back at lab 3, I noticed an issue with my function. Figure 5 above shows the error and figure 6 shows the correct code. I started by adjusting the "for" command so that it read "for student in student_data" and correcting my indentation error with the f-string. These changes on their own did not correct the issue, so I also added a second "print" command at the end so it matched what I did in lab 3 exactly. After running it now, it worked (but I am not entirely sure why). This is something I will continue to tinker with to try to better understand what went wrong and how I fixed it.

```
171     @staticmethod
172     def output_student_courses(student_data: list):
173         """ This function displays the current data in the file to the user
174
175         Changelog (who, when, what)
176         LBatista, 11.19.2023, created function
177
178         return: None
179         """
180         print("-" * 50)
181         for student in student_data:
182             print(f'Student {student["FirstName"]} 'f'{student["LastName"]} is enrolled in {student["CourseNar
183         print("-" * 50)
184         print()
```

Figure 6 - corrected output student courses function

Now I was technically finished with my code for assignment 6, but I wanted to play around a bit more with the formatting on the input student data function. I liked how in assignment 5 I was able to adjust the

input to allow for some special characters (allowing the user to have a hyphenated last name), rather than immediately throwing an error if they didn't use only letters. Figure 7 shows that adjusted code with the "if any" command and the new value error.

```
try:
    student_first_name = input("Enter the student's first name: ")
    if any(char.isdigit() for char in student_first_name):
        raise ValueError("The first name should only contain letters. ")
    student_last_name = input("Enter the student's last name: ")
    if any(char.isdigit() for char in student_last_name):
        raise ValueError("The last name should only contain letters. ")
    course_name = input("Please enter the name of the course: ")
    student_data = {"FirstName": student_first_name,
                    "LastName": student_last_name,
                    "CourseName": course_name}
    students.append(student_data)
```

Figure 7 - Personalization of the input function code to allow for special characters.

My final step was testing my code in Terminal. I changed the directory to the correct folder and ran the code using Python3. Figure 8 shows the outcome with my modification to the input and a successful run.

```
---- 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.
----------------------------------------


Enter your menu choice number: 2
------------------------------------------------
Student Lisa Batista is enrolled in Python 100
Student Ricardo Batista is enrolled in Python 300
Student Bob Baker is enrolled in Python 200
Student Vic Vu is enrolled in Python 100
Student Hero Robinson-Batista is enrolled in Python 100
------------------------------------------------
```

Figure 8 - Running in Terminal

Summary

I am pleased I was able to correct my errors from last week and use JSON effectively in my assignment this week. I also found that my original confusion regarding classes eased as I worked through Lab 03 and the assignment this week. I feel that while I am still a little lost on some of the commands and would not be able to write the code without the help of a starter file, I am getting a better grasp of how it all works and why I need to format the code a certain way. I am building confidence in baby steps– enough that I want to try to add in more modifications and I don't feel discouraged when an error is raised. I think my next step will be attempting this assignment again without the starter file to see if I can write the entire code from scratch on my own. After that, I'd like to try to get good enough that I can do it for memory (but we are certainly not there quite yet).