Name: Luis Candela

Date: 11/20/2024

Course: Foundations of Programming (Python)

Module 06 - Functions

Introduction

In the previous module the concept of dictionaries, JSON files and error handling were covered, giving the student a broader understanding of the advantages of using collections for data analysis and processing.
This module will delve into the topic of code architecture and the use of functions and classes to simplify script into a more comprehensive structure that enables coders to understand the programs workflow, maximize script efficiency and reduce program errors.
Functions are reusable blocks of code that perform tasks and can serve as modules to break down a program into more manageable segments. This will make debugging and editing easier. Functions can work with variables inside or outside of its domain, these are called local or global. Once a function is called for it works with specific variables under set parameters to return a desired outcome.
In order to organize functions, coders use Classes, a larger module made of groups of functions and data which provide a better structural organization. A program will usually has more than one Class, each class focuses on a specific process of the script and combining the work of each all processes the program delivers the product result.
The structure of a program will be related to its function, the desired outcome and the work flow stages necessary to get to an end result. The logic behind this is called the “Separation of Concerns”.
This assignment will teach us how to refactor a script into a more comprehensive set of Classes and Functions that simplifies the code’s length, organizes its’ contents into logical steps and helps manage the data to return same result.

Assignment 06
Create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course. This program is very similar to Assignment05, but it adds the use of functions, classes, and using the separation of concerns pattern.

The script begins exactly like Module05 Assignment. It has a header, imports JSON work module and defines constants/variables.

```
# ------------------------------------------------------------------------------ #
# Title: Assignment06_Starter
```

```
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Luis,11/20/2024,Modified Script
# ------------------------------------------------------------------------------------------ #
import json  # Importing the json module for working with JSON data

# Define the Data Constants (strings and filenames used throughout the program)
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 where the student enrollment data will be stored
FILE_NAME: str = "Enrollments.json"
```

It is important to mention that most of the previous variables will no longer be needed because with functions it is unnecessary to use temporary variables to store data and make better use of code by creating and assigning value to new variables directly inside the functions.
In this section we begin by grouping our functions into two classes:

1. FileProcessor: Contains all functions that process data and reads/writes files.
2. IO: Contains functions that interact and obtain information from the user.

This first class has two functions one which as its name describes reads the initial list of data and the second which writes new data to a file.

```
# Define the Data Variables and constants
students: list = []  # A list that will hold all student data (dictionaries for each student)
menu_choice: str  # Variable to store the menu choice made by the user

# Class for processing file operations (reading from and writing to files)
class FileProcessor:
    @staticmethod
    def read_data_from_file(file_name: str) -> list:
        # Tries to read student data from the specified file and load it into the student_data
list
        try:
```

```
            with open(file_name, "r") as file:  # Open the file in read mode
                student_data = json.load(file)  # Load the data from the file (assumed to be
JSON formatted)
            return student_data  # Return the loaded data
        except Exception as e:  # If there's an error, catch it and display a message
            print("Error: There was a problem with reading the file.")
            print("Please check that the file exists and that it is in a json format.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)  # Print any error documentation
            print(e.__str__())  # Print the string representation of the error
            return []  # Return an empty list if reading fails
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        # Tries to write the student data to the specified file
        try:
            with open(file_name, "w") as file:  # Open the file in write mode
                json.dump(student_data, file)  # Write the student data as JSON to the file
            print("The following data was saved to file!")  # Notify the user that the data has
been saved
            # Print the student data that was saved to the file
            for student in student_data:
                print(f'Student {student["FirstName"]} {student["LastName"]} is enrolled in
{student["CourseName"]}')
        except Exception as e:  # If there's an error, catch it and display a message
            print("Error: There was a problem with writing to the file.")
            print("Please check that the file is not open by another program.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)  # Print any error documentation
            print(e.__str__())  # Print the string representation of the error
```

The second class contains all the functions have some interaction with the user: prints menus, requests information, displays data and prompts error messages or exceptions.

```
# Class for handling input/output operations like displaying the menu and getting user
input
class IO:
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        # Placeholder method to output error messages if needed
        pass
```

```
    @staticmethod
    def output_menu(menu: str):
        # Display the menu to the user
        print(MENU)

    @staticmethod
    def input_menu_choice():
        # Prompt the user to input a choice from the menu and return it
        return input("What would you like to do: ")
```

Classes are a higher order of hierarchy than functions and must be indented properly. It is important to notice that this program works a simple list of data that does not need special treatment nor does it have sub-classes, so it is run by a “@staticmethod” class where functions are called on and execute their tasks without extra layers of code as we will learn in the future called an Instance Class.

```
    @staticmethod
    def output_student_courses(student_data: list):
        # Display the list of students and their course enrollments
        print("-" * 50)  # Print a line for visual separation
        for student in student_data:  # Loop through all students in the list
            print(f'Student {student["FirstName"]} {student["LastName"]} is enrolled in
{student["CourseName"]}')
        print("-" * 50)  # Print another line for visual separation
```

Each function’s synthax can have the ability to set parameters or arguments on what data they will work with in that part of the program. Some functions here have a parenthesis with the arguments (lowercase, snake_case) that are names of variables to be called upon to work in the function. These variables can be global or local; this simplifies the code enormously because the same function code can be reused with different variables by simple typing the new arguments into the function when called for.

```
    @staticmethod
    def input_student_data():
        # Prompt the user to enter a student's details and return the student dictionary
        try:
            student_first_name = input("Enter the student's first name: ")  # Get first name
from user
            if not student_first_name.isalpha():  # Check if the first name contains only letters
                raise ValueError("The first name should not contain numbers.")
            student_last_name = input("Enter the student's last name: ")  # Get last name
from user
            if not student_last_name.isalpha():  # Check if the last name contains only letters
```

```
            raise ValueError("The last name should not contain numbers.")
        course_name = input("Please enter the name of the course: ")  # Get course
name from user
        student_data = {"FirstName": student_first_name, "LastName":
student_last_name, "CourseName": course_name}
        return student_data  # Return the student dictionary
    except ValueError as e:  # If a ValueError occurs, print the error message
        print(e)
        print("-- Technical Error Message -- ")
        print(e.__doc__)  # Print any error documentation
        print(e.__str__())  # Print the string representation of the error
    except Exception as e:  # For any other error, print a generic error message
        IO.output_error_messages("Error: There was a problem with your entered data.",
e)
```

Once the classes and functions are defined the main body of the script simple calls on them to execute the desired tasks. As in the previous version of this program, we run a while loop of user-oriented menu choices to input, display and store data in a list of dictionary rows.
In this case the user's menu choice calls upon a function from a certain class to execute the task; the main body is reduced to only the functions and the programs output in each step.
Before the loop the program loads data from the specific FILE_NAME (list) and continues to build on it.

```
# Read data from file at the start of the program (loads student data if available)
students = FileProcessor.read_data_from_file(FILE_NAME)

# Main loop: presents the menu and processes user input until the user decides to exit
while True:
    IO.output_menu(MENU)  # Display the menu
    menu_choice = IO.input_menu_choice()  # Get the user's choice from the menu

    if menu_choice == "1":  # If the user selects option 1 (Register a Student)
        student_data = IO.input_student_data()  # Get the new student data(no parameter)
        if student_data:  # If the student data was successfully entered
            students.append(student_data)  # Add the new student to the students list
            print(f"You have registered {student_data['FirstName']}
{student_data['LastName']} for {student_data['CourseName']}.")
        continue  # Return to the start of the loop for another choice

    elif menu_choice == "2":  # If the user selects option 2 (Show current data)
        IO.output_student_courses(students)  # Display the current list of students and
their courses
```

```
        continue  # Return to the start of the loop for another choice

    elif menu_choice == "3":  # If the user selects option 3 (Save data to a file)
        FileProcessor.write_data_to_file(FILE_NAME, students)  # Write the data to the file
        continue  # Return to the start of the loop for another choice

    elif menu_choice == "4":  # If the user selects option 4 (Exit the program)
        break  # Exit the loop and end the program

    else:  # If the user selects an invalid option
        print("Please only choose option 1, 2, or 3")  # Prompt the user to select a valid option

print("Program Ended")  # Print a message when the program ends
```

The code ends when the user selects menu choice 4 and exits the loop.

Summary

In this module we learned to break down a script into comprehensive modules using the "Separation of Concerns". With classes and functions our previous code can be compartmentalized into reusable blocks of code that eliminate the need to repeat code or add intermediate variables. Defining empty variables to store data is no longer required because functions can work with built-in local variables to process data.

Classes group functions into an understandable work-flow of back- end data processing and front-end user interface features.

The main body of the program is simplified into function "calls" that can be used repeatedly throughout the code with different variables.

Refactoring greatly improves code work-flow and reduces programming errors.