Rabiya Wasiq

11/19/2023

Introduction to Programming with Python

Assignment06

FUNCTIONS

Introduction

This week we were introduced to the concept of functions in Python programming. We started with the concept of global and local variables and using them in functions. We them moved on to adding parameters to our functions and passing arguments to those parameters when calling the function. We then moved on to classes and learned about static method and instance method. We also learned to organize our code based on concerns.

Creating Python Script

For the purpose of this assignment I will be organizing my script from Assignment 06 into functions based on concerns, namely processing and presentation. I will then de defining functions for each class with parameters. I will also demonstrate passing arguments into my functions as I call them later in my program script.

Functions

A function is a block of code which only runs when it is called. They are very useful for repeated set of commands and help in writing more modular programs. Function facilitate code reusability, we can add parameters to a function and later pass arguments to those parameters when the function is called. By doing do we can use the same function for different arguments.

Classes

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

(External link - https://docs.python.org/3/tutorial/classes.html)

Separation of concern

Separation of concerns is a principle used in programming to separate an application into units, with minimal overlapping between the functions of the individual units. The separation of concerns is achieved using modularization, encapsulation and arrangement in program layers.

Starter Code from Assignment05

We start by using Open Tab in Pycharm and navigating to the location of Python script of Assignment 05. We will copy the script and make changes as we proceed.

We will use the same format of adding a script header and using pseudocode for problem solving.

```
# ------------------------------------------------------------------------
-------------- #
# Title: Assignment06
# Desc: This Assignment Demonstrates Functions and Separation of Concern
# Change Log: (Who, When, What)
#   Rabiya Wasiq,11/18/2023,Created Script
#   Rabiya Wasiq 11/19/2023, Updated I0 function getting_student_data and
FileProcessor function writing_data_to_file
# ------------------------------------------------------------------------
-------------- #
```

Figure 1 – Script Header

Declaring and assigning Variables / Constants

Declaring and assigning variables are the first steps to writing a code. It is best practice to declare the variables and constants that you intend to use throughout the script at the beginning.

```
# Define the Data Constants
MENU: str = '''
---------------------------------------
---- Course Registration Program ----
  Select from the following menu:
    1. View all students registered to date
    2. Register a New Student for a Course
    3. Show New student registration details
    4. Save New student data to a file
    5. Exit the program

---------------------------------------
'''
import json
from json import JSONDecodeError
from typing import TextIO


# Define the Data Variables
FILENAME: str = 'Enrollments.json'
```

```
menu_choice: str = ''
new_student: dict = {}
students: list[dict[str,str]] = []
```

Figure 2: Declaring and assigning constants and variables.

In (**Figure 2)** I have defined the constants and variables that I intend to use in this program. Since we will be reading data from a Json file I start the code by importing Json, which helps me load data directly into variables with minimal formatting needed.

Class: FileProcessor

I want to divide my program into two areas, processing and presentation.

For processing I start by creating a class FileProcessor, this class will include a function that can read data from Json file, store them to a variable for the code and also a function that can write data to Json file.

```
#--------------Processing-----------------

class FileProcessor:

    @staticmethod
    def read_data_from_file( File_Name : str) ->list[dict[str,str,str]]:
        """
        This function reads data from Json file and stores it into a list of
dictionaries
        :param File_Name:
        :return: student_data:list[dict[str.str,str]]
        """
        File_Name : str
        student_data : list[dict[str,str,str]] = []
        file :TextIO = None
        try:
            file = open(File_Name, 'r')
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_message('Json file not found, creating it...',e)
            file = open(File_Name, 'w')
        except JSONDecodeError as e:
            IO.output_error_message('Json file does not contain any data,
resetting it..',e)
            file = open(File_Name, 'w')
            json.dump(student_data, file)
        except Exception as e:
            IO.output_error_message('Unexpected Technical error',e)
        finally:
            if not file.closed:
                file.close()
        return student_data
```

Figure 4 – Function FileProccessor.read_data_from_file

In (**Figure 4**)I have defined a function that can read data from Json file, I have added a parameter to the function which is a list of dictionaries. The function also has a return value which would help me save the return value from this function into a variable that I can use later in the program. It is important to note that this function truncates the file, hence I have added parameters to pass in the variables that can contain both data read from the Json file and also new data received from the user.

```
@staticmethod
def writing_data_to_file(student_row: dict,
students_data:list[dict[str,str,str]]):
    """
    Writes data to Json file in the format list of dictionaries.
    :param student_row:
    :param students_data:
    :return:
    """
    student_row: dict
    students_data: list[dict[str, str, str]]

    if student_row["First_Name"] == '' or student_row["Last_Name"] == '' or
student_row["Course_Name"] == '':
        IO.output_message('Please enter student details again')
    else:
        students_data.append(
            student_row)

    try:
        file = open(FILENAME, 'w')  # using the write function, to truncate
the file
        json.dump(students, file)
        file.close()
        IO.output_message('Student registration details recorded\n')
    except Exception as e:
        IO.output_error_message('Unexpected Technical error',e)
    finally:
        if not file.closed:
            file.close()
```

Figure 5 – FileProcessor. writing_data_to_file

In **(Figure 5)** I have defined a function that first saves the new data input received by the user to a list of dictionaries, and then loads that list of dictionaries to a Json file. The parameters to this function is dictionary that should contain the new data, a list of dictionaries that stores all data and name of the Json file.

Class : I0 (Input / Output)

For presentation I start by creating a class IO which stands for Input / Output. I will use this class to bundle all functions that would prompt any message to the user and subsequently collect input from the user.

```
#--------------Presenting----------------------
class IO:

    @staticmethod
    def output_error_message(message : str, error:Exception=None):
```

```
        """ This function displays a custom error messages to the user
        default exception value set to none
          :return: None
        """
        print(message, end="\n\n")

        if error is not None:
            print("-- Technical Error Message -- ")
            print('-' * 50)
            print(error, error.__doc__, error.__str__(), type(error),
sep='\n')


    @staticmethod
    def output_message(message : str):
        """ This function displays a custom messages to the user
          :return: None
        """
        print(message, end="\n\n")
```

Figure 6 – IO.output_error_message and IO.output_message

In **(Figure 6)** I have created the function output_message that would print a message to the user. We can pass the message as argument to the function. The function output_error_message is a similar function with two parameters, message and error. The default value of error is set to None, with an if condition also added. If the value of error is not none the function will print additional statements.

```
@staticmethod
def output_menu(menu: str):
    """
    This function displays the menu options to the user
    :param menu:
    :return: None
    """
    print(menu)
```

Figure 7 – IO.output_menu

In **(Figure 7)** the function output_menu prints the menu to the user.

```
@staticmethod
def input_menu_choice():
    """
    This functions gets the menu choice from the user
    :return: menu_choice
    """
    menu_choice: str
    try:
        menu_choice = input("Enter your menu choice number: ")
        if menu_choice not in ("1", "2", "3", "4","5"):
            raise Exception("Please enter the correct menu choice")
    except Exception as e:
        IO.output_error_messages(e)
    return menu_choice
```

Figure 8 – IO. input_menu_choice

In ***(Figure 8)*** the function input_menu_choice gets menu choice as input from the user and also returns the string value entered by the user. I have also wrapped it in an exception which prompts an error message based on an If condition.

```
@staticmethod
def current_data_from_file(student_data:list[dict[str,str,str]]) ->str:
    """
    This function displays all student data from the Json file, formatted in a string
    :param student_data:
    :return:
    """
    student_data: list[dict[str, str, str]]
    student_row :dict
    for student_row in student_data:
        IO.output_message(f'Student Full Name :{student_row["First_Name"]}
{student_row["Last_Name"]} | Course Name : {student_row["Course_Name"]}')
```

Figure 9 – IO. current_data_from_file

In **(Figure 9)** the function current_data_from_file presents data from a list of dictionaries to the user using string formatting and for loop. Later in the code we will be passing the variable that stored the data read from the Json file as an argument to this function.

```
@staticmethod
def input_student_data() -> dict:
    """
    This function gets first name, last name, course name from the user and adds them to a
dictionary
    :param student_data:
    :return: student_row
    """
    student_row :dict = {}
    student_first_name: str = ''
    student_last_name: str = ''
    course_name: str = ''

    while True:
        try:
            student_first_name = input("Enter the student's first name: ").capitalize()
            if not student_first_name.isalpha():
                raise ValueError
            break
        except ValueError:
            IO.output_error_message('Student First Name can only contain alphabetic characters')
            #Not passing error deatils
    while True:
        try:
            student_last_name = input("Enter the student's last name: ").capitalize()
            if not student_last_name.isalpha():
                raise ValueError
            break
        except ValueError:
            IO.output_message('Student Last Name can only contain alphabetic characters')

    course_name = input("Enter the course name: ")
    student_row = {"First_Name": student_first_name, "Last_Name": student_last_name,
                   "Course_Name": course_name}
    return student_row
```

Figure 10 – IO. input_student_data

In **(Figure 10)** the function input_student_data prompts the user to enter first name, last name and course name. I have added exceptions to ensure that the student enters the details in the correct format. I have used IO.output_error_message function to prompt the error message, I have not passed in the exception as I

do not want to present it to the user. The function then adds all the input received from the user to a dictionary and also returns the dictionary.

```
@staticmethod
def present_student_data(student_row : dict):
    """
    This function presents data from a dictionary to the user in string formatting
    :param student_row:
    :return:
    """
    student_row:dict
    if student_row["First_Name"] == '' or student_row["Last_Name"] == '' or
student_row["Course_Name"] == '':
        IO.output_message('Please enter student details again')
    else:
        message = f'{student_row["First_Name"]} {student_row["Last_Name"]} has
registered for {student_row["Course_Name"]}'
        IO.output_message(message)
```

Figure 11 – IO.present_student_data

In (**Figure 11)** the function present_student_data extracts data from a dictionary by accessing key values and formats it into a string.

```
@staticmethod
def exit_choice()-> str:
    """
    This function presents the user the choice to exit the program
    :return: exit_choice
    """
    exit_choice: str = ''
    exit_choice = input("Do you wish to exit the program? Y/N").capitalize()
    return exit_choice
```

Figure 12 – IO.exit_choice

In (**Figure 12)** the function exit_choice confirms if the user wants to exit the program and returns the input value.

Main body of the code

After defining the functions at the start of the code, we can now move on to the main body of the code. We will execute our code by calling the functions that we have defined and pass in the relevant arguments. We will be storing the return values to code's variables as we progress.

We start by reading data from the Json file using the function FileProcessor.read_data_from_file and passing in the constant "FILENAME".**(Figure 12)**The function returns the data as a list of dictionaries which I store in the variable "students"

```
#Reading data from file and loading to global variable students
students = FileProcessor.read_data_from_file(File_Name=FILENAME)
```

Figure 12 – Reading data from JSON file

```
# Present and Process the data
while True:
    IO.output_menu(menu=MENU)   # Present Menu
    menu_choice = IO.input_menu_choice()

    # Menu choice 1 shows the data extracted from the JSON and saved in the two-
dimensional list
    if menu_choice == '1':
        students = FileProcessor.read_data_from_file(File_Name=FILENAME)
        IO.current_data_from_file(student_data=students)

    # Getting student details from the user
    elif menu_choice =='2':
        new_student = IO.input_student_data() #storing new student data as a
dictionary to be used later in the program

    #presenting new student registration details to the user
    elif menu_choice =='3':
        IO.present_student_data(student_row=new_student)

    #writing new student details to Json file
    elif menu_choice =='4':

FileProcessor.writing_data_to_file(student_row=new_student,students_data=students,File
_Name=FILENAME)

    #exiting the program
    elif menu_choice == '5':
        exit_choice = IO.exit_choice()
        if exit_choice == "Y":
            IO.output_message('\nPausing the program till you press Enter...\n')
            break
```

Figure 13 – While Loop

I then start a while loop that would prompt the menu to the user using the IO.output_menu function and passing in the constant MENU and stores the return value in a variable menu_choice.

I then start adding my if conditions.

If menu_choice is 1, I present the data read from the JSON file to the user using the IO.current_data_from_file function and passing in the variable “students”.

If menu_choice is 2, I prompt the user to enter details using the IO.input_student_data and store the return value in the form of a dictionary names “new_student”.

If menu_choice is 3, I present the student details received using the IO.present_student_data and passing in the variable ”new_student” (that contains the student details from menu_choice 2)

If menu_choice is 4, I write the student details received to a Json file, using the FileProcessor.writing_data_to_file function. Since this function truncates the file we pass in the variables "new_student' (student details received from the user), "students"(data read from the Json file at the start of the program) and "FILENAME" (name of the file where we want to write the data)

If menu_choice is 5, I present the user with the choice to exit the program using the IO.exit_choice function

Summary

I was successfully able to create classes and define functions with parameters. I was able to organize my code on the basis of concerns. For the main body of the code I was successfully able to call functions and pass in arguments.

Calling functions made the body of the code much clean and simple to manage. I particularly found passing in arguments to functions very useful. The return values were also very helpful in storing values from the functions into variables that can be used later in the code.