Alejandro Guillen

November 20, 2023

IT FDN 110 B

ASSIGNMENT 06: Functions

Introduction

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it repeatedly.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Parameters:
A parameter is the variable defined within the parentheses during function definition. Simply they are written when we declare a function.

Arguments:
A parameter is the variable defined within the parentheses during function definition. Simply they are written when we declare a function.

In programming, variables can be categorized as either local or global, depending on where they are declared and where they can be accessed.

Local Variables: Variables declared within a function are considered local to that function. These variables can only be accessed and used within the function where they are defined. They are "inside of the function's scope.

Global Variables: Variables declared outside of any function, typically in the main body of the script, are considered global to the entire script. These variables can be

accessed and modified from anywhere within the script. They are "in scope" throughout the entire script.

Classes and Functions

A class is a code template for creating objects. Objects have member variables and have behaviors associated with them. In python a class is created by the keyword class. An object is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances, assigning the class to a variable.

Step 1: Define Libraries, constants, variables.

First, it was to define the libraries JSON, constants, variables and organize the code according to the template for this module and define the input and outputs. The variables were set to empty string, student_data for row was set as dictionary.

```
import json
from json import JSONDecodeError

# 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.
    4. Exit the program.
-----------------------------------------
'''
FILE_NAME: str = 'Enrollments.json'
student_table: list = []
```

Step 2: Class FileProcessor

The program includes a class named FileProcessor, inside this class I define read_data_from_file() and write_data_to_file(). When the program starts, the program opens the file: "Enrollments.json".

If the file exists is automatically read and then held into the list of dictionary student_data. The program opens File_NAME object using the open function in mode read. Also, I have added the exception handled in this part, due to the object

FILE_NAME should exist before and it must have data. If the file does not exist, so will catch the exception FileNotFoundError or another exception, it will run the function: IO.output_error_messages and then will create the file as well. This function read_data_from_file() return student_data.

The function write_data_to_file() opens the file in mode write, use dump() function to save the data from students list to file and close. Also, each student row from students_data list is returned and displayed into choice 3 (menu_choice)

```
# Processing --------------------------------------- #
class Fileprocessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    AGuillen,11.19.2023,Created Class

    """

    # When the program starts, read the file data into a list of lists (table)
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            print('The file data was loaded successfully')
        except FileNotFoundError as e:
            IO.output_error_messages("Text file must exist before running this
script!", e)
            # file = open(file_name, 'w')
            # json.dump(student_data, file)  # Loading initial data [] from
collection students
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            print("The following data was saved to file!")
            for student in student_data:
                print(f'Student {student["FirstName"]} '
                      f'{student["LastName"]} is enrolled in
{student["CourseName"]}')
        except TypeError 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)
        finally:
            if not file.closed:
                file.close()
```

Step 3: ***Class IO***

The program includes a class named IO, inside this class I define output_error_messages () to handle errors, output_menu() to display the menu, input_menu_choice() to input the choice number, add_data_to_table() to input the student to student_data list and output_current_data() to display the current data.

```
# Presentation ------------------------------------ #
class IO:
    """
    A collection of presentation layer functions that manage user input and
output

    ChangeLog: (Who, When, What)
    AGuillen,11.19.2023,Created Class
    AGuillen,11.20.2023,Added menu output and input functions
    AGuillen,11.21.2023,Added a function to display the data
    AGuillen,11.22.2023,Added a function to display custom error messages
    """

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

        ChangeLog: (Who, When, What)
        AGuillen,11.26.2023,Created 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')

    @staticmethod
    def output_menu(menu: str):
        print(menu)
        print()


    @staticmethod
    def input_menu_choice():
        """ This function gets a menu choice from the user

        :return: string with the users choice
        """
        choice = "0"
        try:
```

```
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"):
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())
        return choice


    @staticmethod
    def add_data_to_table(student_data: list):
        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.append({"FirstName": student_first_name, "LastName":
student_last_name, "CourseName": course_name})
            print(f"You have registered {student_first_name} {student_last_name}
for {course_name}.")
        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_current_data(student_data: list):
        print("-"*50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in
{student["CourseName"]}')
        print("-"*50)
        return student_data
```

Step 4: While loop:

This while loop allows the user to choose for the different options according to the menu:

Choice 1: Runs IO.input_menu_choice () to add a new student and enroll in a course.

Choice 2: Runs IO.output_current_data () to display the current students enrolled.

Choice 3: Runs Fileprocessor.write_data_to_file () to add a new student and enroll in a course.

Choice 4: break out the while loop and display the message "Program Ended".

```
while True:
    # Present the menu of choices
    IO.output_menu(menu=MENU)
    # Input user data
    menu_choice = IO.input_menu_choice()
    if menu_choice == "1":
        student_table = IO.add_data_to_table(student_data=student_table)
        continue
    # Present the current data
    elif menu_choice == "2":
        # Process the data to create and display a custom message
        student_table = IO.output_current_data(student_data=student_table)
        continue
    # Save the data to a file
    elif menu_choice == "3":
        student_table = Fileprocessor.write_data_to_file(file_name=FILE_NAME,
student_data=student_table)
        continue
    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Summary

- I have learned in this module about how to make an application more flexible to other applications and keep our code clear and reusable. This module has been interesting due to the handling of errors properly using try-except blocks, I have used functions with arguments and classes to process, input and output the data. Also, I have learned some basic concepts for programming like:

- Encapsulation is a fundamental concept that involves bundling data and the processes that work on that data into a single "capsule" of code. It's like putting related information and actions together in one place for better organization and control. In simpler terms, encapsulation is like packing related stuff in a box, keeping it safe from outside interference, and providing a way to interact with it as needed. Functions with return values encapsulate their logic, making it easier to test, debug, and maintain. You can test the function's output independently without relying on its internal state or side effects.

- Flexibility: By using parameters, you can call the function with different file names and student data, making it more flexible and reusable in various contexts.
- Reusability: Functions with return values are more reusable because they don't rely on or modify external state. You can use them in various contexts without worrying about unexpected changes to global data.
- Clarity and Predictability: Return values make it clear what the function computes or produces. When you see a function that returns a value, you immediately know what to expect from it. This enhances code readability and helps other developers understand your code.
- Using parameters is a better practice because it promotes encapsulation, reusability, and predictable behavior in the functions. It also reduces the reliance on global data, which can lead to cleaner and more maintainable code. The version of a function that takes parameters is almost always preferable over the one using global variables.