Jocelyn Munoz

August 4, 2024

Foundations of Programming: Python

Assignment 06

https://github.com/Joce9322/IntroToProg-Python-Mod06

Functions and Classes

Introduction

This document reviews how to incorporate functions and classes into our program for increased organization, reusability, and compartmentalization. The first section will cover how to write and call functions, and how to integrate functions into our student registration program to perform multiple tasks repeatedly. The second section will review the benefits of grouping functions together within classes. As our program becomes more complex, the dynamic between functions and classes will streamline the management and maintenance of code.

The program continues to build off the previous assignments, and the defined constants are still the same. Most of our variables, however, will be used locally within a function, rather than globally. Please reference the previous assignments for a review of the program variables, constants, and inputs/outputs.

Functions

Functions are a way to perform a task multiple times in different places throughout our program without having to rewrite the same code repeatedly. In essence, functions group code into reusable blocks (Matthes 2023). First, to define a function, we start by typing the function header, starting with def followed by the function name, parenthesis (), and a colon: that marks the end of the function header. Inside the pair of parentheses, we can include values described as parameters. After the colon, on the next indented line, we start the function body. The function body includes the tasks the function will provide. To execute the code inside a function, we need to call it by typing out the function name followed by parenthesis and any arguments being passed.

Parameters and Arguments

A variable inside a function's parentheses is referred to as a parameter. For example, in *Figure 1*, the function read_data_from_file has two parameters: file_name: str and students: list. A parameter serves as a piece of information the function needs to perform the tasks inside the body (Matthes 2023). An argument is a piece of information that is passed from a function call to a function (*Figure 2*). Thus, when we call the function, after the function has been defined, we place the value (argument) we want the function to use.

```
@staticmethod
def read_data_from_file(file_name: str, student_data:list):
    """
        This function reads student registration data from a json file

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created Function

        :param file_name: string for the file name
        :param student_data: list of dictionary rows containing student registration content
        :return student_data
    """
```

Figure 1. The components of the function read_data_from_file include two parameters.

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 2. The arguments “FILE_NAME” and “students” are passed in the function call.

The added benefit of using parameters in your function is getting to use variables locally rather than globally. Global variables are defined before the function and outside of the function’s parameters or body. Even though global variables can be accessed within functions and anywhere in the program, the use of global variables introduces potential side effects. Any changes made to the global variables will affect their values globally and can cause maintainability, reusability, and readability issues as programs become longer and more complex.

Returns

A function is also capable of returning a value or set of values after the data has been processed rather than using print() statements to display the output (*Figure 3*). A returned value can be saved in a variable to be modified or used later in the program. However, return values allow for the results of a function to be output immediately without having to save it in a variable and this can be especially convenient if the return value won’t be used again.

```
def read_data_from_file(file_name: str, student_data:list):
    """
        This function reads student registration data from a json file

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created Function

        :param file_name: string for the file name
        :param student_data: list of dictionary rows containing student registration content
        :return student_data
    """
    try:
        file = open(FILE_NAME, "r")
        student_data = json.load(file)
        file.close()
    except FileNotFoundError as e:
        IO.output_error_messages( message: "Text file must exist before running this script!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
    return student_data
```

Figure 3. The function read_data_from_file returns the student_data list after the data has been read from the json file.

Classes

A class allows us to group functions together to improve the organization and maintenance of code by separating the program into areas of concern. For example, in our program we have two main responsibilities or areas of concern: file processing and input/output processing.

After writing a class, we can create objects from the class through instantiation. For this program, however, we are creating static classes in which the function code can be called on the class and not on an instance of the class. To do

this, we use the @staticmethod decorator before defining the function. To write a class, we use a similar approach to defining functions:

class ClassName:
    @staticmethod
    def function_name():
        body of function

Modifications to our program

Assignment 06 modifies our program to include two classes and seven functions. We created a FileProcessor class that holds a collection of functions to work with JSON files. The two functions in the FileProcessor class are read_data_from_file and write_data_to_file (*Figure 4*).

```python
@staticmethod
def read_data_from_file(file_name: str, student_data:list):
    """
        This function reads student registration data from a json file

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created Function

        :param file_name: string for the file name
        :param student_data: list of dictionary rows containing student registration content
        :return student_data
    """
    try:
        file = open(FILE_NAME, "r")
        student_data = json.load(file)
        file.close()
    except FileNotFoundError as e:
        IO.output_error_messages( message: "Text file must exist before running this script!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
    return student_data
```

```python
@staticmethod
def write_data_to_file(file_name: str, student_data:list):
    """
        This function writes student registration data to a json file

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created Function

        :param file_name: string for the file name
        :param student_data: list of dictionary rows containing student registration content
    """
    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( message: "Please check that the data is a valid JSON format", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 4. Code defining the read_data_from_file and write_data_to_file functions.

Our second class, IO, holds a collection of five functions that manage user input and output for student registration. The first function, output_error_messages, is used for error handling and displays custom error messages to the

user. It contains two parameters: a message string for a custom error message and an optional parameter for the technical error message, its documentation, and type (*Figure 5*).

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

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created Class
    """

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

        ChangeLog: (Who, When, What)
        JMunoz,8.3.2024,Created function

        :param message: string displaying custom error messager
        :param Exception: exception to print
        :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 5. Code defining the output_error_messages function.

In the IO class, we also have a function to output_menu, which holds on parameter for a menu string. This function prints the menu of choices to the user. Then we have an input_menu_choice function that gets a menu choice from the user (*Figure 6*). The input_student_data function is used to get the first name, last name, and course name from the user. Lastly, the output_student_courses function outputs the current list of registered students for their courses (*Figure 6*).

```
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

    ChangeLog: (Who, When, What)
    JMunoz,8.3.2024,Created function

    :param menu: string displaying the menu message
    :return: None
    """
    print()
    print(menu)
    print()  # Adding extra space to make it look nicer.

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

    ChangeLog: (Who, When, What)
    JMunoz,8.3.2024,Created function

    :return: string with the users choice
    """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1","2","3","4"):  # Note these are strings
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())  # Not passing the exception object to avoid the technical message

    return choice
```

Figure 6. Code defining the output_menu and input_menu_choice functions in the IO class.

The main body of the script still contains our while loop for menu choices 1-4, but the code looks much cleaner because it calls on the defined classes and functions, and passes arguments to the function parameters (*Figure 7*).

```python
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeat the following tasks
while True:
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    if menu_choice == "1": #Register a student for a course
        students = IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2": #Display registered students
        IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "3": #Save data to file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4": #Exit program
        break #out of the while loop

print("Program Ended")
```

Figure 7. Code in the main body of the script that calls on functions and passes arguments.

Summary

In this assignment, we improved the organization and readability of our code to include classes and functions for student registration. We were able to write new functions, minimize global variables, and pass arguments in the main body of the script. We also wrote classes to separate functions based on the program's main areas of concerns. Rather than working with objects, we used static classes that allow for functions to be called directly on the class. The current structure and organization of our program makes it easier to debug sections of code at a time. As our program continues to expand, maintaining simple blocks of code will be more efficient than debugging messy, complicated, and repetitive lines of code.