Nicole Tsao
November 25, 2024
IT FDN 100A
Assignment 06

The Role of Functions and Classes in Code Organization

Introduction:

Module 6 of the class focused heavily on organizing your code. We are introduced to functions, parameters, and classes. These tools are used to make code more modular, organized, reusable, and easier to maintain.

- Modular – Breaking down a program into smaller, manageable, and reusable components.
- Organized – Code becomes easier to structure, often by segregation of different functionalities.
- Reusable – Once defined, classes and functions can be used in multiple places without having to rewrite the same logic.
- Easier to maintain - By organizing code into logical components, it's easier to update, debug, and extend the codebase.

Let's go over Assignment 06 and how we successfully utilized all organization tools that were discussed. We were also able to visualize how much of a difference they really make.

1.) Nothing really changes with how you start writing your script. We continue to follow the same practice with a short description and documentation using a header, we import the external modules or libraries needed for the current script, and most importantly: **<u>we define our constants and variables.</u>**

```python
# ------------------------------------------------------------------------------------------
# Title: Assignment06
# Desc: This assignment demonstrates using functions
# with structured error handling
# Nicole Tsao, 11/25/2024, Created Script
# ------------------------------------------------------------------------------------------ #
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.
    4. Exit the program.
----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.

```

2.) We can start creating our classes. You can think of how you want the functions segregated or what is called the ***separation of concerns***. In this assignment we divide them into the processing class which we labeled as class FileProcessor and the presenting class which we labeled as class IO.

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

        Ntsao,11.25.2024,Created Class
        Ntsao,11.25.2024,Added menu output and input functions
        Ntsao,11.25.2024,Added a function to display the data
        Ntsao,11.25.2024,Added a function to display custom error messages
    """
```

Figure 2.1 It is good practice to create and add docstrings to your classes, especially when collaborating with other developers.

```python
class FileProcessor:
    """
        A collection of processing layer functions that work with Json files

        Ntsao, 11/25/2024, Created script
    """
    pass
```

Figure 2.1 Use “pass” as a placeholder while the class is still empty to avoid getting any errors when you want to run and test out your code.

3.) Begin defining your functions and make sure they are defined under the correct class. (Let’s briefly go over parts of a function)

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

        Ntsao,11.25.2024,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")
```

Figure 3.1

1.) def – keyword: Defines a function. All functions begin with “def”
2.) output_error_messages – the function name
3.) (message: str, error: Exception = None) – the parameters that the function accepts
4.) This represents the function body, code to perform an action

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

        Ntsao,11.25.2024,Created Class
        Ntsao,11.25.2024,Added menu output and input functions
        Ntsao,11.25.2024,Added a function to display the data
        Ntsao,11.25.2024,Added a function to display custom error messages
    """

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

            Ntsao,11.25.2024,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")
```

Figure 3.2 This is how the functions would look like once defined in the class. Note that we use @staticmethod – a method that belongs to a class but does not require access to the class or instance (self)

4.) Create the body of your script and start using or calling the functions that you have created. You may also start using the functions in defining other functions as well.

```python
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "r")

            student_data = json.load(file)

            file.close()
        except Exception as e:
            IO.output_error_messages( message: "Error: There was a problem with reading the file.", e)

        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 4.1 Here we define the function read_data_from_file and utilize the function that we Previously defined as output_error_messages.

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

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(students)
        continue

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

Figure 4.2 Main body of the script calling all the functions that we created in our classes

Summary:

Using classes and functions to organize code is a great way to keep things clean, manageable, efficient, and easy to scale. Classes help you group related data and behaviors together, making it easier to model real-world objects and keep your code organized. This way, you avoid clutter and repetition. Functions are perfect for breaking down complex tasks into smaller, reusable chunks. They make your code more modular, so you can test, debug, and update parts of your program independently. Ending this with a before and after of how a script can look like when utilizing classes and functions.

From this:

```
73      elif menu_choice == "2":
74          try:
75              student_first_name=(input("What is the student's first name? "))
76              if not student_first_name.isalpha():
77                  raise ValueError("The first name should not contain numbers.")
78              student_last_name=(input("What is the student's last name? "))
79              if not student_last_name.isalpha():
80                  raise ValueError("The last name should not contain numbers.")
81              try:
82                  student_gpa=float(input("What is the student's GPA? "))
83              except ValueError:
84                  raise ValueError("GPA must be a numeric value.")
85
86
87              student_data={"FirstName":student_first_name,
88                        "LastName":student_last_name, "GPA":student_gpa}
89              students.append(student_data)
90          except ValueError as e:
91              print(e)
92              print("--Technical Error found--")
93              print(e.__doc__)
94              print(e.__str__())
95          except Exception as e:
96              print("There was a non-specific error")
97              print("--Technical Error Message")
98              print(e, e.__doc__, type(e), sep="\n")
99
100             continue
```

To this:

```
168     if menu_choice == "1":  # This will not work if it is an integer!
169         students = IO.input_student_data(students)
170         continue
```