Mariam Assaad
Wednesday, November 20, 2024
IT FDN 100 A - Foundations of Programming: Python
Assignment 06

Exploring Functions, Arguments, Returns, Classes, and Programming Best Practices

Introduction

This document serves as a comprehensive guide to key programming concepts that improve the structure, readability, and efficiency of code. It explores fundamental topics like functions, arguments, return values, classes, and the separation of concerns, which are essential for writing clean, maintainable, and reusable programs. By understanding and applying these principles, developers can enhance the quality of their work and simplify the process of applications.

1. Using Functions

This section introduced the concept of using functions to improve code organization, readability, and reusability. By breaking a program into smaller, self-contained functions, developers can make their code easier to debug and maintain. Key tasks involved reorganizing an existing script from the previous module into reusable functions, defining global variables where needed, and implementing error handling.

Implementation in Assignment 6:

- Functions like read_data_from_file(), output_menu(), and write_data_to_file() demonstrate breaking tasks into smaller, reusable components.
- Error handling is integrated into functions to catch specific issues.\

```
def read_data_from_file(file_name: str, student_data: list):
    """Reads data from a file and loads it into a list of
dictionaries."""
    try:
        with open(file_name, "r") as file:
            student_data.extend(json.load(file))
    except FileNotFoundError as e:
        IO.output_error_messages("The file does not exist. Starting
with an empty list.", e)
    except json.JSONDecodeError as e:
        IO.output_error_messages("Error decoding JSON. Starting with an
empty list.", e)
    except Exception as e:
        IO.output_error_messages("An unexpected error occurred.", e)
```

Figure 1: Function for reading data from a file, demonstrating modularity and error handling.

2. Using Arguments

This section delved into the concept of parameters and arguments, explaining how they make functions more flexible and reusable. Parameters act as placeholders for the values (arguments) passed to a function during its call. This section highlighted the importance of parameters in customizing a function's behavior and reducing the dependency on global variables.

Implementation in Assignment 6:

- Functions use arguments like file_name and student_data to operate independently of global variables.
- Named arguments are used for clarity in function calls.

```
FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
```

Figure 2: Example of using arguments in function calls for better flexibility and reusability.

3. Using Returns

This section explains how functions can return values instead of changing external variables. Returning values makes functions easier to understand, reuse, and maintain.

Key Learnings:

- **What Return Values Do**:
  - Functions can send back results using the return statement.
  - These results can be saved in variables for later use or used immediately.
- **Why Use Returns**:
  - **Clarity**: Makes it clear what the function does and what it outputs.
  - **Reusability**: Functions work independently and can be reused in different situations.
  - **Safety**: Prevents accidental changes to data outside the function.
  - **Error Handling**: Functions can return errors or success messages for better debugging.
- **How It Works**:
  - Functions can return one or more values.
  - Returned values can be reused multiple times without running the function again.

Example Behavior:

- **Immutable Objects** (e.g., numbers or strings): Changing them inside a function doesn't affect them outside.
- **Mutable Objects** (e.g., lists or dictionaries): Changes inside a function affect the original data unless handled carefully.

Implementation in Assignment 6:

While functions like read_data_from_file() modify student_data directly, the structure ensures clarity and predictability by encapsulating processing logic within the function.

```
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """Reads data from a file and loads it into a list of dictionaries."""
    try:
        with open(file_name, "r") as file:
            student_data.extend(json.load(file))
    except Exception as e:
        IO.output_error_messages("An unexpected error occurred.", e)
```

Figure 3: Static method example showing predictable behavior with encapsulated logic.

4. Using Classes

This section focuses on organizing code with classes, which group functions, variables, and constants. Classes provide a modular structure, making code easier to manage, especially in larger projects.

Key Learnings:

1. **Purpose of Classes**:
    - Classes group related data and functions into one logical unit.
    - Functions inside classes are often called methods.
2. **Benefits of Classes**:
    - **Modularity**: Makes code more organized and easier to understand.
    - **Reusability**: Objects created from classes can operate independently.
    - **Encapsulation**: Classes can hide details and expose functionality.
3. **Static Methods**:
    - Functions that do not rely on instance-specific data.
    - Declared with @staticmethod.
    - Can be called without creating an object of the class.
4. **Use of self**:
    - Represents the instance of the class being operated on.
    - Used in methods to access instance-specific data.
    - Not used in static methods.
5. **Adding Documentation with Docstrings**:
    - Provides quick reference to the purpose of classes and methods.

- Integrated development environments (IDEs) display docstrings as tooltips.
- Example: Add clear documentation for functions like add() or subtract() to explain their parameters and return values.

Implementation in Assignment06:

- The FileProcessor and IO classes group file handling and input/output operations, respectively.
- Static methods like @staticmethod make it clear that the functions don't depend on class instances.

```
class FileProcessor:
    """A class to handle file processing operations."""

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        ...

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        ...


class IO:
    """A class to handle input and output operations."""

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

    @staticmethod
    def input_menu_choice() -> str:
        return input("Please select a menu option (1-4): ")
```

Figure 4: Classes for organizing file and I/O operations in a modular structure.

5. Separation of Concerns

The Separation of Concerns is a key design principle that organizes code into different sections based on their specific tasks. This helps make code cleaner, easier to understand, and simpler to maintain.

Key Learnings:

1. **What is a Concern?**
   - A concern is a specific job or responsibility in the program, like showing data to users, processing data, or storing it.
2. **Three Types of Concerns**:
   - **Presentation**: Manages how data is shown to users and handles user input/output.

- **Logic**: Handles the main tasks or calculations of the program.
- **Data Storage**: Deals with storing, retrieving, and managing data.

3. **Why Separate Concerns?**
   - **Easier to Read and Modify**: You can change one part of the code without affecting others.
   - **Reusable Code**: Each section can be reused in other programs.
   - **Fewer Bugs**: Reduces mistakes when updating or fixing code.
   - **Better Teamwork**: Teams can work on different sections at the same time.
4. **How to Apply it**:
   - Divide code into three parts:
     - **Data**: Define variables and constants.
     - **Processing**: Write functions or calculations.
     - **Presentation**: Handle input from users and display results.
5. **Benefits**:
   - Keeps your code organized and easy to understand.
   - Makes it easier to update or scale different parts of the program.

Testing and Running Assignment 6

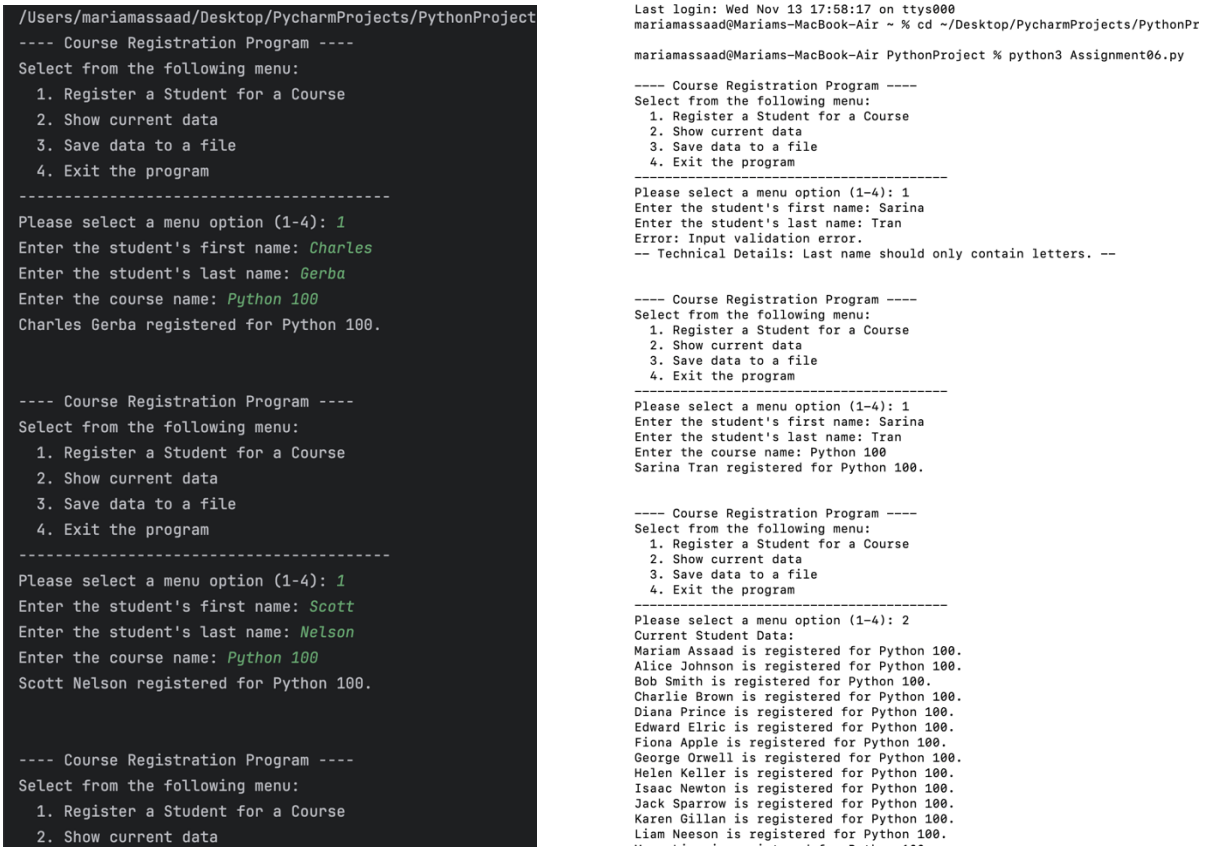

Figure 5: The program runs correctly in both PyCharm and from the console or terminal.

Also, I created a repository for Assignment 05, where I uploaded all the code and files from this module. You can view the repository and project files here:

https://github.com/mariam-assaad/IntroToProg-Python-Module06

Summary

This document explores essential programming concepts that enhance the structure, maintainability, and functionality of code. It begins with functions, emphasizing their role in breaking down complex scripts into reusable and modular components while incorporating error handling and global variables where needed. It then delves into arguments, explaining how parameters make functions flexible and reusable by processing input without relying on global variables. The use of return values is highlighted for improving clarity, reusability, and immutability by avoiding direct modifications to external variables. The document also covers classes, which organize related data and methods into modular units, enhancing code scalability, encapsulation, and readability, especially in large projects. Finally, it discusses the Separation of Concerns principle, which divides code into distinct layers—Data, Logic, and Presentation—to simplify updates, reduce bugs, and encourage collaboration. Together, these principles promote the development of clean, efficient, and professional code, essential for building robust and scalable software systems.