**Mia Bolding**

07/31/2024

Introduction to Programing with Python

Assignment 06– Programming Basics

GitHub Link

# Python Functions and Classes

## Introduction

This week in module 6, I learned about the basics of programming with functions and classes. I learned to differentiate between functions, parameters, and arguments and their role in organizing code. I explored return values, global and local scopes, and their interaction with functions. I also dove into classes, understanding how they differ from functions and help structure code.

## New Definitions

**Functions:** are reusable block of codes that perform a task or set of tasks. Two important purposes of functions are modularity and reusability. I can use functions to break down any large program into smaller and easier to manage pieces. Because I can easily understand small parts of the program and maintain or debug them separately. The second useful feature of function is that I can define and name a certain function once and reuse them throughout the program. This saves time and makes programs leaner.

In order to use functions, use the def keyword followed by the function name and parentheses (). Inside the parentheses, I can specify parameters. See below for function syntax:

For example:

**def function_name(parameters):**

**# code block return value**

**# optional**

**Classes:** class in Python are like templates for making objects. These objects can have their own data (called attributes) and actions (called methods) that work with the data. To make a class, I use the class keyword, then the name of the class, and a colon.

For example:

**def __init__(self, parameters):**

# constructor method self.attribute = value def method_name(self, parameters):

# method code block return value  # optional

In short, functions are like tools for doing specific tasks, while classes are like blueprints for making objects that hold both information and actions.

**Parameters:** are like placeholders I put inside the parentheses when defining a function. They hold the spot for values I'll pass to the function later. Using parameters lets the function take in different inputs, making it more flexible and reusable.

For example:

**def add_numbers(a, b):**

    # 'a' and 'b' are parameters

**Arguments**: are the actual values I provide to a function when I call it. They are used to replace the parameters defined in the function.

**def add_numbers(a, b**):  # 'a' and 'b' are parameters

    **return a + b result = add_numbers(5, 3)**

    # 5 and 3 are arguments print(result)

    # Output: 8

In the above example, add numbers is the function. a and b are parameters that the function uses to accept input. 5 and 3 are arguments passed to the function when it is called, allowing the function to add these two numbers and return the result.

## Lesson Learn During Assignment

The script has two classes with a total of eight functions. The **FileProcessor** class has two functions: *read_data_from_file()* to read data from a JSON file and *write_data_to_file()* to write data to a JSON file. The **IO class** includes six functions: *output_menu()* displays the menu to the user, output_error_messages() shows error messages, *input_menu_choice()* gets the user's menu choice, *output_student_and_course_name()* presents student and course information, and *input_student_data()* collects and validates student details. The main body of the script handles the primary loop and control flow, guiding the program's execution.

Figure 1: **FileProcessor** Class, read and write functions

```python
# Processing ------------------------------------- #
class FileProcessor:
    """ A collection of processing layer functions that work with JSON files
        ChangeLog:
        MBolding, 8/9/2024 Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list) -> list:...
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):...
```

Figure 2. The **IO class** and its functions

```python
# Presentation ----------------------------------- #
class IO:
    @staticmethod
    def output_menu(menu):...  1
    """A collection of presentation layer functions that manage user input and output
    ChangeLog:
    MBolding, 8/9/2024 Created Class
    MBolding, 8/9/2024 Added output menu that I forgot
    """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...  2

    @staticmethod
    def input_menu_choice() -> str:  3
        """This function gets a menu choice from the user
        :return: string with the user's choice
        """
        choice = "0"
        try:...
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Not passing `e` to avoid the technical message

        return choice

    @staticmethod
    def output_student_and_course_name(student_data: list):...  4

    @staticmethod
    def input_student_data(student_data: list) -> list:...  5
```

## Summary

I initially found this assignment overwhelming, but looking back, I see how using classes and functions made the program more organized and manageable. For example, by using the FileProcessor class to handle file operations and the IO class to manage user interactions, I was able to keep different parts of the program separate. This separation made it easier to focus on individual tasks and reduced complexity.

Specifically, the FileProcessor.read_data_from_file() function allowed me to neatly handle reading data from JSON files, while the IO.input_student_data() function helped in gathering and validating user input. This structure not only improved readability but also made debugging and modifying the program simpler. By breaking the code into these well-defined components, I was able to handle the overall task more effectively, demonstrating how classes and functions can make complex assignments more manageable.