Chad Conklin

11/23/2023

IT FDN 110 B Au 23: Foundations Of Programming: Python

Assignement 07

<https://github.com/cconk/uwpython>

Python Programming

# Introduction

# In the ever-evolving world of software development, understanding the core concepts of programming languages like Python is crucial for developers. Python, known for its simplicity and readability, is a popular choice for beginners and experienced programmers alike. It offers a robust set of features that cater to a wide range of applications, from web development to data science. To fully leverage the power of Python, it's essential to grasp the foundational elements that constitute its structure and functionality.

# At the heart of Python programming are key concepts such as statements, functions, and classes, each playing a unique role in the construction of efficient and effective code. Furthermore, as object-oriented programming (OOP) forms a significant part of Python, understanding the nuances of different types of classes, along with the functionalities like constructors, attributes, properties, class inheritance, and method overriding, is paramount. These elements not only enhance code organization and reusability but also pave the way for more advanced programming techniques. Additionally, tools like Git and GitHub Desktop are integral to modern coding practices, especially in collaborative environments, and understanding their differences and uses is vital for any developer.

# Topic 1

With these considerations in mind, let’s delve into some specific questions that shed light on these fundamental aspects of Python and related tools:

# Differences between Statements, Functions, and Classes:

# Statements: These are the basic units of code that Python executes, like a line in a script. Examples include assignment (a = 5), conditional (if), loop (for), etc.

# Functions: Functions are blocks of reusable code that perform a specific task. They can accept inputs (parameters), execute code, and return a result. For example, def my\_function():.

# Classes: Classes are blueprints for creating objects (instances). They encapsulate data (attributes) and behavior (methods) into one single unit. For instance, class MyClass:.

# Difference Between a Data Class, a Presentation Class, and a Processing Class:

# Data Class: Primarily focused on storing data. Its main purpose is to hold data and possibly implement some basic logic around this data.

# Presentation Class: Designed for the user interface and presentation logic. It handles how data is displayed and interacted with on the user interface.

# Processing Class: Concerned with processing or manipulating data. This includes business logic, data processing, and other complex operations.

# What is a Constructor?

# A constructor is a special method in a class used to initialize new objects. In Python, it is defined as \_\_init\_\_(self, ...). It sets up the state of a new object by initializing attributes.

# What is an Attribute?

# An attribute is a variable that is bound to an instance of a class or to the class itself. Instance attributes are specific to each instance, while class attributes are shared across all instances of the class.

# What is a Property?

# A property in Python is a special kind of attribute that allows for getter, setter, and deleter methods. This enables encapsulation and allows for attributes to be accessed and modified in a controlled manner (like through a getter and setter).

# What is Class Inheritance?

# Class inheritance is a feature in object-oriented programming where a new class (child or subclass) inherits attributes and methods from an existing class (parent or superclass). This allows for code reusability and hierarchical relationships between classes.

# What is an Overridden Method?

# An overridden method is a method in a subclass that has the same name, parameters, and return type as a method in its superclass, but with a different implementation. It's used to provide specific behavior in the subclass, while still maintaining the interface of the superclass.

# Difference Between Git and GitHub Desktop:

# Git: It is a distributed version control system used for tracking changes in source code during software development. It is command-line tool and is widely used for its flexibility and power in handling various software development tasks.

# GitHub Desktop: It is a Graphical User Interface (GUI) for Git, specifically tailored for GitHub. It simplifies using Git for version control and collaboration by providing a user-friendly interface, making it easier for those who prefer not to use command-line tools.

# Topic 2

For the final task included in the assignment document concerning creating a program using constants, variables, and print statements, I developed and successfully ran the following program both in the command prompt and IDLE. Additionally, I successfully used Visual Studio Code to develop the solution. The following code, including the comments therein, encompasses the solution I was able to provide. These are the steps I completed and focus points based on the starter code:

1. Script Header: Updated the script header with my name and current date.
2. Constants: Several of the constants were already defined like MENU and FILE\_NAME as constants, in the starter code.
3. Variables: Similar to the constants several of the variables were defined int the starter code as well like menu\_choice, students, and other necessary variables. These are used within functions and classes.
4. Classes and Functions: Create two classes, FileProcessor and IO, each with their own methods. Here's a brief outline:
   * FileProcessor: This class handles file operations like reading and writing data.
   * IO: This class handles input/output operations like displaying menus and capturing user inputs.
5. Implementing Functions:
   * output\_error\_messages(message: str, error: Exception = None): This function will print error messages.
   * output\_menu(menu: str): This function will display the menu.
   * input\_menu\_choice(): This function captures and returns the user's menu choice.
   * output\_student\_courses(student\_data: list): This function prints the current student data.
   * input\_student\_data(student\_data: list): This function captures new student data from the user.
   * read\_data\_from\_file(file\_name: str, student\_data: list): This function reads data from a file.
   * write\_data\_to\_file(file\_name: str, student\_data: list): This function writes data to a file.
6. Using the @staticmethod decorator
   * The @staticmethod decorator in Python is used to define a method within a class as a static method, which means that it can be called on the class itself without needing an instance of the class. This has several implications and uses:
     1. Instance Independence: A static method does not have access to the instance (self) of the class or any of its instance attributes. It behaves like a regular function but is enclosed within the class's namespace.
     2. Class State Independence: Static methods also do not have access to the class attributes (cls) unless explicitly passed. They are independent of the state of the class.
     3. Usage:
        1. Static methods are used when some functionality is related to the class but does not need to access or modify the class's state.
        2. They are often used to create utility functions within a class.
     4. Syntax and Calling:
        1. Defined using the @staticmethod decorator above the method definition.
        2. Can be called using either the class name or an instance of the class, e.g., ClassName.static\_method() or class\_instance.static\_method().
     5. Comparison with Class Methods:
        1. Unlike static methods, class methods take a cls parameter that points to the class and not the instance when the method is called. They are defined with the @classmethod decorator.
        2. Class methods can modify the class state that applies across all instances of the class, while static methods cannot.

# ------------------------------------------------------------------------------------------ #

# Title: Assignment06\_Starter

# Desc: This assignment demonstrates using functions

# with structured error handling

# Change Log: (Who, When, What)

#   RRoot,1/1/2030,Created Script

#   Chad Conklin,11/14/2023, Modified script to use functions and structured error handling

# ------------------------------------------------------------------------------------------ #

import json

import os

from typing import IO

# 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"

# Data Variables

students: list = []  # a table of student data

#Classes

class FileProcessor:

    """Handles file processing tasks"""

    @staticmethod

    def read\_data\_from\_file(file\_name: str, student\_data: list):

        """ Reads data from a file into student\_data list """

        if not os.path.exists(file\_name):

                # File does not exist, so we start with an empty list

                return

        try:

                with open(file\_name, "r") as file:

                    # Ensure the file is not empty

                    file\_content = file.read()

                    if file\_content:

                        student\_data.extend(json.loads(file\_content))

                    else:

                        # File is empty, so we start with an empty list

                        return

        except json.JSONDecodeError as e:

                IO.output\_error\_messages("Error decoding JSON from file", e)

        except Exception as e:

                IO.output\_error\_messages("Error reading file", e)

    @staticmethod

    def write\_data\_to\_file(file\_name: str, student\_data: list):

        try:

            with open(file\_name, "w") as file:

                json.dump(student\_data, file)

        except Exception as e:

            IO.output\_error\_messages("Error writing to file", e)

class IO:

    """ Handles Input/Output tasks """

    @staticmethod

    def output\_error\_messages(message: str, error: Exception = None):

        print(message)

        if error:

            print("-- Technical Error Message --")

            print(error)

    @staticmethod

    def output\_menu(menu: str):

        print(menu)

    @staticmethod

    def input\_menu\_choice() -> str:

        return input("What would you like to do: ")

    @staticmethod

    def output\_student\_courses(student\_data: list):

        print("-" \* 50)

        for student in student\_data:

            print(f'Student {student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}')

        print("-" \* 50)

    @staticmethod

    def input\_student\_data() -> dict:

        try:

            first\_name = input("Enter the student's first name: ")

            if not first\_name.isalpha():

                raise ValueError("The first name should not contain numbers.")

            last\_name = input("Enter the student's last name: ")

            if not last\_name.isalpha():

                raise ValueError("The last name should not contain numbers.")

            course\_name = input("Please enter the name of the course: ")

            return {"FirstName": first\_name, "LastName": last\_name, "CourseName": course\_name}

        except ValueError as e:

            IO.output\_error\_messages(str(e))

            return {}

# Main processing

FileProcessor.read\_data\_from\_file(FILE\_NAME, students)

while True:

    IO.output\_menu(MENU)

    menu\_choice = IO.input\_menu\_choice()

    if menu\_choice == "1":

        student\_info = IO.input\_student\_data()

        if student\_info:

            students.append(student\_info)

            print(f"You have registered {student\_info['FirstName']} {student\_info['LastName']} for {student\_info['CourseName']}.")

    elif menu\_choice == "2":

        IO.output\_student\_courses(students)

    elif menu\_choice == "3":

        FileProcessor.write\_data\_to\_file(FILE\_NAME, students)

        print("Data saved to file.")

    elif menu\_choice == "4":

        break

    else:

        print("Please choose a valid option (1-4).")

print("Program Ended")

# Summary

In conclusion, I continued to learn a lot about Python throughout this module and I am looking forward to the rest of class.