Chad Conklin

11/23/2023

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

Assignement 07

https://github.com/cconk/IntroToProg-Python-mod07

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:

1. 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:.

2. 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.

3. 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.

4. 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.

5. 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).

6. 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.

7. 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.

8. 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. Here's a summary of the steps taken to complete the script for the Python course registration program:

1. **Defining Constants and Variables:**
    - Defined **MENU** and **FILE_NAME** as constants.
    - Initialized **students** as an empty list to store student data and **menu_choice** as an empty string for user input.
2. **Creating Data Classes:**
    - Developed a **Person** class with properties for **first_name** and **last_name**, including getters and setters with validation.
    - Created a **Student** class that inherits from **Person**, adding a **course_name** property with getters and setters.

3. **Implementing Processing Class:**
    - Designed a **FileProcessor** class with two static methods:
        - **read_data_from_file**: Reads data from a JSON file and loads it into the **students** list, converting JSON objects to **Student** instances.
        - **write_data_to_file**: Writes the **students** list to a JSON file, converting **Student** objects to dictionaries.
4. **Developing Presentation Class:**
    - Constructed an **IO** class for handling user input and output. This class includes methods for displaying the menu, error messages, student data, getting user menu choices, and collecting student registration data.
5. **Incorporating Error Handling:**
    - Added try-except blocks in methods of **FileProcessor** and **IO** for structured error handling, ensuring robustness against file access issues and invalid user inputs.
6. **Main Script Logic:**
    - Loaded initial data from the file into the **students** list at the start of the program.
    - Implemented a loop to display the menu and process user input:
        - Menu option 1: Register a student by getting data from the user and adding it to the **students** list.
        - Menu option 2: Display current student data.
        - Menu option 3: Save current student data to the file.
        - Menu option 4: Exit the program.
7. **Testing and Validation:**
    - Ensured that the program handles different user inputs correctly, reads from and writes to the file as expected, and properly utilizes class properties and methods.

By following these steps, the script effectively manages course registration data, offering functionalities for registering students, displaying registered data, saving data to a file, and handling various errors gracefully.

```
# ------------------------------------------------------------------------------
----------- #
# Title: Assignment07
# Desc: Demonstrates using data classes with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Chad Conklin, 11/23/2023, Modified for Assignment07 requirements
```

```python
# -------------------------------------------------------------------------------
----------- #
import json

# 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 = []  # Table of student data
menu_choice: str = ""  # User's menu choice

# Data Classes
class Person:
    """ Represents a person """
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self) -> str:
        return self._first_name

    @first_name.setter
    def first_name(self, value: str):
        if not value.isalpha():
            raise ValueError("First name must contain only letters")
        self._first_name = value

    @property
    def last_name(self) -> str:
        return self._last_name

    @last_name.setter
    def last_name(self, value: str):
        if not value.isalpha():
            raise ValueError("Last name must contain only letters")
        self._last_name = value

    def __str__(self) -> str:
        return f"{self.first_name} {self.last_name}"

class Student(Person):
    """ Represents a student, inheriting from Person """
    def __init__(self, first_name: str, last_name: str, course_name: str = ""):
        super().__init__(first_name, last_name)
        self.course_name = course_name

    @property
    def course_name(self) -> str:
        return self._course_name

    @course_name.setter
    def course_name(self, value: str):
        self._course_name = value

    def __str__(self) -> str:
        return f"{super().__str__()} enrolled in {self.course_name}"

# Processing Class
class FileProcessor:
    """ Processes data to and from a file """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ Reads data from a file and populates the student_data list.
            If the file does not exist, creates a new empty file. """
        try:
            with open(file_name, "r") as file:
                student_data.clear()
                data = json.load(file)
                for item in data:
                    student_data.append(Student(item["first_name"],
item["last_name"], item["course_name"]))
        except FileNotFoundError as e:
            IO.output_error_messages("File not found. Creating a new file.", e)
            FileProcessor.write_data_to_file(file_name, student_data)  # Create
an empty file
        except Exception as e:
            IO.output_error_messages("An error occurred while reading the file.",
e)

    @staticmethod
```

```python
    def write_data_to_file(file_name: str, student_data: list):
        """ Writes the student_data list to a file. """
        try:
            with open(file_name, "w") as file:
                json.dump([{"first_name": student.first_name, "last_name":
student.last_name, "course_name": student.course_name} for student in
student_data], file)
        except Exception as e:
            IO.output_error_messages("An error occurred while writing to the
file.", e)
# Presentation Class
class IO:
    """ Handles input and output operations """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        print(message)
        if error is not None:
            print("-- Technical Error Message --")
            print(error, error.__doc__, type(error), sep='\n')

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

    @staticmethod
    def input_menu_choice() -> str:
        choice = input("Enter your menu choice number: ")
        return choice

    @staticmethod
    def output_student_and_course_names(student_data: list):
        print("-" * 50)
        for student in student_data:
            print(student)
        print("-" * 50)

    @staticmethod
    def input_student_data() -> Student:
        first_name = input("Enter the student's first name: ")
        last_name = input("Enter the student's last name: ")
        course_name = input("Enter the course name: ")
        return Student(first_name, last_name, course_name)

# Main Body of Script
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 = IO.input_student_data()
        students.append(student)

    elif menu_choice == "2":
        IO.output_student_and_course_names(students)

    elif menu_choice == "3":
        FileProcessor.write_data_to_file(FILE_NAME, students)

    elif menu_choice == "4":
        break

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.