Erik Duong

December 2, 2024

Course: Python IT FDN 110

Assignment: 07

https://github.com/erikduong807/IntroToProg-Python-Mod06

Create Course Registration Program

Introduction

This document outlines the steps taken to create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course. This program will **add the use of set of data classes.**

1. Object-Oriented Design

Two primary classes were introduced:

- **Person Class**: Handles first_name and last_name attributes with integrated validation via properties.

```python
class Person:
    def __init__(self, first_name: str = "", last_name: str = ""):
        self._first_name = first_name
        self._last_name = last_name

    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        '''
        Returns the first_name as a title
        :return: the first name, properly formatted
        '''
        return self._first_name.title()  # formatting code

    @first_name.setter
    def first_name(self, value: str):
        '''
        Sets the first name, while doing validations
        :param value: the value to set
        '''
        if value.isalpha() or value == "":  # is character or empty string
            self._first_name = value
        else:
            raise ValueError("The first name should not contain numbers or special characters.")
```

- **Student Class**: Inherits from Person and adds a course_name attribute for specialized functionality.

```python
class Student(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):
        '''
        Returns the course name as a title
        :return: the course name, properly formatted
        '''
        return self._course_name.title()  # formatting code

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

    def __str__(self):
        '''
        The string function for Student
        :return: the string as a csv value
        '''
        return f'{super().__str__()},{self.course_name}'
```

This design facilitated data encapsulation and reuse, reducing redundancy and improving maintainability. Structured objects replaced dictionary handling, centralizing logic and ensuring consistent behavior across the program.

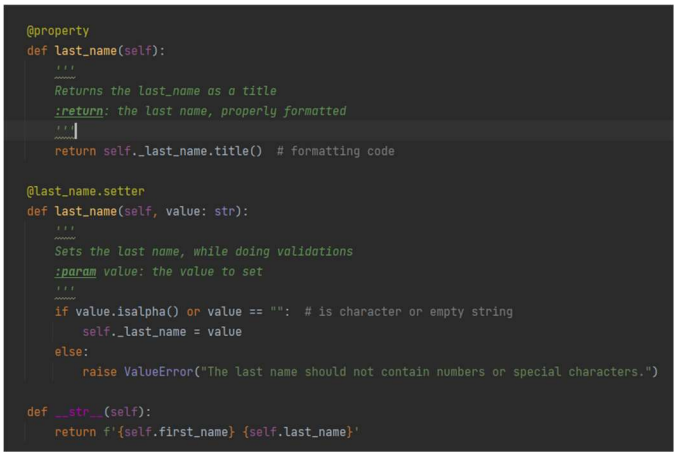
2. Modular Program Structure

The program was divided into three main components:

- **Person and Student Classes**: Focused on data encapsulation and validation.
- **FileProcessor Class**: Managed JSON file operations, ensuring object-to-dictionary conversion and vice versa.
- **IO Class**: Handled user interaction, including menu display, error messages, and input collection.

3. Property

In my code, properties are used in Person and Student to validate input (e.g., ensuring first_name contains only alphabetic characters).



```python
@property
def last_name(self):
    '''
    Returns the last_name as a title
    :return: the last name, properly formatted
    '''
    return self._last_name.title()  # formatting code

@last_name.setter
def last_name(self, value: str):
    '''
    Sets the last name, while doing validations
    :param value: the value to set
    '''
    if value.isalpha() or value == "":  # is character or empty string
        self._last_name = value
    else:
        raise ValueError("The last name should not contain numbers or special characters.")

def __str__(self):
    return f'{self.first_name} {self.last_name}'
```

These concepts were foundational to improving the code structure, readability, and maintainability in Assignment 07. By applying:

- **Statements** to control the program flow.
- **Functions** for modular operations.
- **Classes** to encapsulate data and behavior.

I was able to create a well-organized program. The use of **properties** for validation, **inheritance** to reduce redundancy, and potential **method overriding** highlight the power of OOP principles in building robust and scalable applications.