Sonali R. Mishra

June 3 2025

Foundations of Programming: Python

Assignment for Module 7

https://github.com/codebeaker/IntroToProg-Python-Mod07/

Assignment 7: Objects, Properties, Inheritance

Introduction

In this module we learned how Python creates objects, and the syntax for getting and setting properties. We also learned how classes of objects can be set up to inherit properties from parent classes. In the assignment we convert our code to use student and person objects, and move properties to the objects. We can also keep much of the validation at the property level, improving the readability of the code (at least to fellow developers).

Objects and Properties

Objects are instances of classes. When defining a class, you can define the **properties**, or specific attributes, that each instance of the class (each object) should have. Using classes and objects means that you can have multiple instances of an object in the same program. For instance, if you are programming a game with multiple horses, you may want to create multiple horse objects that all have specific properties (for instance, they may all appear on the screen the same way, or have different gaits as they move about the screen).

In this assignment we shift from using lists of dictionaries to creating student objects. The Person object has the first and last name properties:

```
class Person:
    '''
    A class representing person data.
    Properties:
    - first_name (str): first name.
    -last_name (str): last name
    '''
    #initialize the person object
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name


    #get/set first_name property
    @property
    def first_name(self):
        return self.__first_name.title()
    @first_name.setter
    def first_name(self, value: str):
        if value.replace(" ","").isalpha() or value =="":
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    #get/set last_name property
    @property
    def last_name(self):
        return self.__last_name.title()

    @last_name.setter
    def last_name(self, value: str):
        if value.replace(" ", "").isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")
    def __str__(self):
        return f"{self.first_name},{self.last_name}"


        def __str__(self):
            return f"{self.first_name},{self.last_name}"
```
Figure 1: Person object

For each property, the code under the @property decorator retrieves the value, while the code under the decorator ending with ".setter" sets the property value. This latter code chunk also contains some validation. In this case we have specified that the first and last name can have spaces and alphabetic characters, but not numbers.

Inheritance

In this module we learned about **inheritance**, meaning when the properties associated with one class are passed down to another class. In this assignment we do this with the class student, which inherits first and last name from the person class but adds another property, course name.

```
class Student(Person):
    """
    A class representing student data. Inherits first name and last name
    from Person and has an additional property, course_name.
    Properties:
    -first_name (str): first name [inherited]
    -last_name (str): last name [inherited]
    -course_name (str): course name.
    """

    def __init__(self, first_name: str = '', last_name: str = '',
                 course_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name
    # get/set course_name property
    @property
    def course_name(self):
        return self.__course_name.title()

    @course_name.setter
    def course_name(self, value: str):
        # allow alphanumeric characters
        if value.replace(" ", "").isalnum() or value == "":
            self.__course_name = value
        else:
            raise ValueError("The course name should not contain special
characters.")

    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```
Figure 2: Student object

The course_name property is set to allow alphanumberic characters and spaces, as course names may be made up of several words and have numbers in them.

Note that we need to convert saved data from a list of dictionaries to student objects, then convert it back to dictionaries when we write data to the file. The write_data_to_file function is included below in Figure 3 as an illustration.

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes data to a json file with data, first
     converting a list of objects to a list of dictionary
     rows and then writing to json

    :param file_name: string data with name of file to write to
    :param student_data: list of student objects to be writen to the file

    :return: None
    """

    try:
        students_saved_dict: list = [] #list to hold the dictionary rows

        for student in student_data:
            student_as_dict: dict = {"FirstName": student.first_name,
                                     "LastName": student.last_name,
                                     "CourseName": student.course_name}
            students_saved_dict.append(student_as_dict)

        file = open(file_name, "w")
        json.dump(students_saved_dict, file)
        file.close()

    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Please check that the file is not open by another
program."
        IO.output_error_messages(message=message,error=e)
    finally:
        if file.closed == False:
            file.close()
```
Figure 3: Function to write data to file, including code for converting data from student objects to a list of dictionaries.

As with other assignments I have written the program to display saved and unsaved students separately, to help the user see what has not yet been saved. The program moves unsaved data to saved data when the user chooses the appropriate menu option. However this is a separate function from the write_data_to_file function, to keep the code modular. Both functions are called when the user selects mneu option 3.

```
# Save the data to a file
elif menu_choice == "3":
    IO.move_unsaved_to_saved(students_unsaved, students_saved)
    FileProcessor.write_data_to_file(FILE_NAME, students_saved)
    #show the user the data that is now saved as confirmation
    IO.output_student_and_course_names(students_saved, students_unsaved)
    continue
```
Figure 4: Calling more than one function when menu option 3 is selected

Summary
In this module we learned about objects and properties. In this assignment we switch from using lists of dictionaries to using student objects, which have properties for first name, last name, and course name. We experiment with inheritance by creating both a person class with first name and last name properties that the student class inherits. Because we are using objects instead of dictionaries, we convert to and from dictionaries when reading and writing to/from a JSON file.