Name: Thomas Perkins

Date: November 30, 2023

Course: IT FDN 110 Foundations of Programming: Python

Assignment: Module 07

Classes and Objects

1.0 Introduction

Classes and Functions were introduced in Module 06, as well as a discussion on their use in modular reusable code. Derived classes and inheritance were brought up but not discussed in detail. In this Module, classes are more fully described and their relation to Object Oriented Programming discussed. Derived classes are utilized instead of just being introduced, and the code from the Module 06 Assignment is updated to change from using lists of strings for data processing and storage to using lists of objects that are defined by object classes.

2.0 Object Classes

In Module 06 it was mentioned that Classes could be instantiated, that is set to a local variable, where all of the attributes (parameters) of that class would come over as well. These instances become objects in the program memory space. As an Object-Oriented language (although it did not start out that way), Python creates parameters as methods of object classes.

An effective way to set these parameters is using getter and setter pairs. In Python the getter function uses the @property decorator, which allows for the creation of a class object parameter (method). The *@param*.setter decorator then can set the initial value of the parameter, formatting, error handling, etc. Both the getter and the setter lines use the *self* object. In Python *self* refers to the instance of the class (itself), in other words the prime instance. Using self in getter and setter code allows for clearer and more reliable parameter management.

```python
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):
    return self.__first_name.title()  # formatting code

@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__first_name = value
    else:
        raise ValueError("The last name should not contain numbers.")
```

Figure 1: Object Class Parameter Setup Using Getter/Settter Pair

Technically these objects are functions within the classes (since they use the *def* command to initiate), however the operate differently than the static method functions shown in Module 06. Although it is possible to mix object functions and static method functions in the same class, from a code organization perspective it would be preferable to separate the two (or at least put them in sub-classes).

3.0 Class Inheritance in Python

Python refers to a class made from another class as a *derived class,* although the terms child class and subclass are frequently used as well. Subclasses are created using the *Derived(Source)* syntax. Any functions (including objects) are inherited from the source class, after which they made be modified and/or new functions defined in the subclass.

The advantage of using derived classes is that inheritance allows for the effective partitioning and reuse of parameters. If similar (or the same) parameters were defined separately in a number of classes, desynching becomes very likely over time (if not inevitable), which will quickly lead to errors, particularly if the use of these parameters is spread over several code modules. Using sub-classes enables a clear pedigree of parameter sources and properties, which will stabilize the code for the project.

In Python the __init__ function is a built-in function of a created class and allows for initialization of the class as an object. When using a derived class, the properties of the source class are “fetched” by initiating the source class while initiating the new derived class, as shown in Figure 2.

```python
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
```

Figure 2: Initiating a Derived Class

4.0 Assignment

The Assignment for the Module was to further modify the code for the Student Registration Program that has been developed from the beginning of the class by creating and using Object Classes. A parent Class, *Person* is defined with *first_name* and *last_name* parameters, and a child of that class *Student(Person)* is defined, inheriting *first_name* and *last_name* and adding the *course_name* parameters. Objects are instantiated using *Student(Person)* and used to create the student data list, for error handling, and for .json input/output functions.

4.1 Results

Please see attached .py file and/or https://github.com/UWtsperk/mod07 for the completed file. Figures 1 and 2 show "happy path" results in PyCharm and the Command Console respectively, Figure 3 shows the final *Enrollments.json* file. Figure 4 show a new error handling function included for this Module that detects duplicate entries in the dataset.

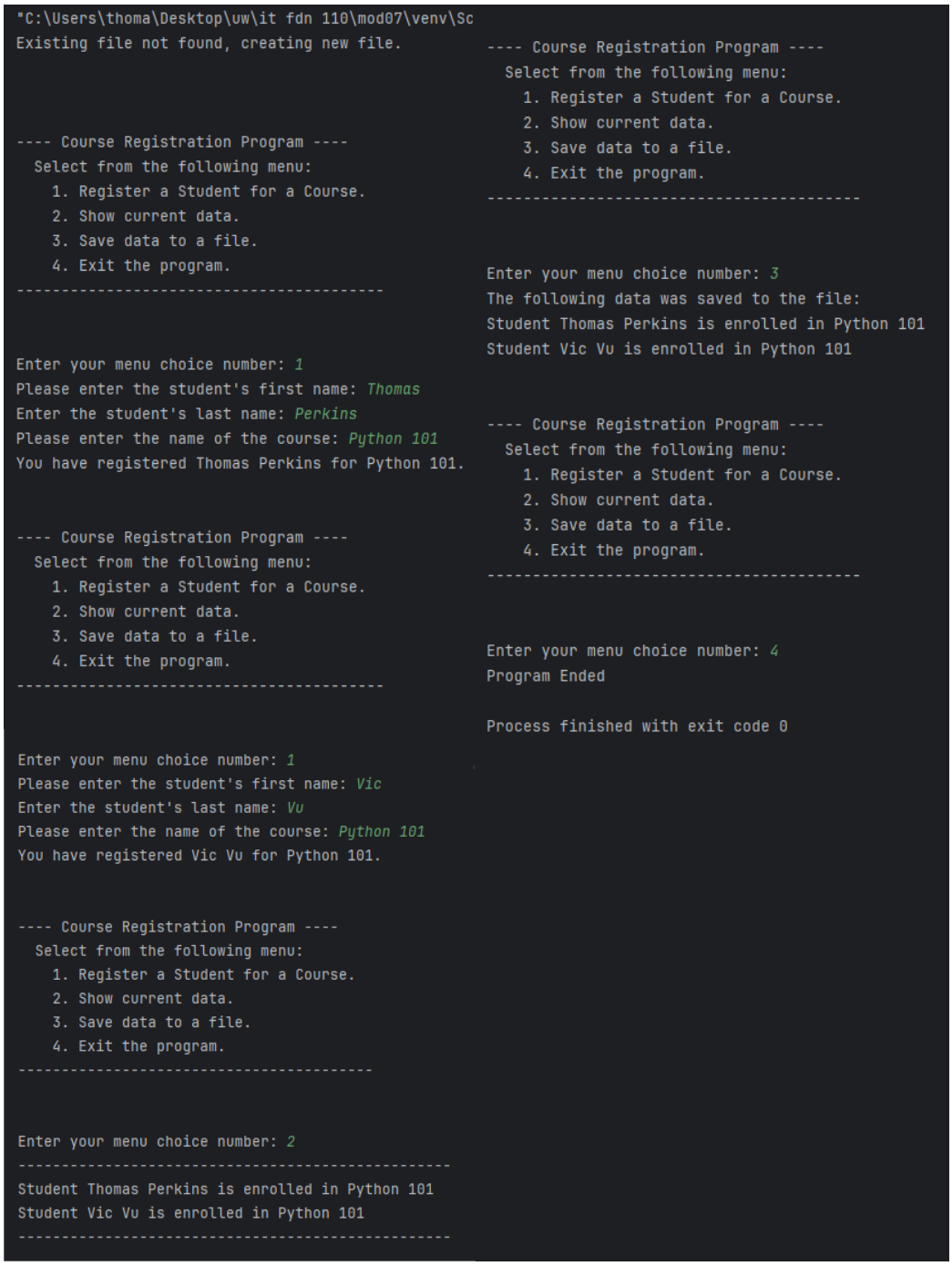

Figure 3: PyCharm Console Output

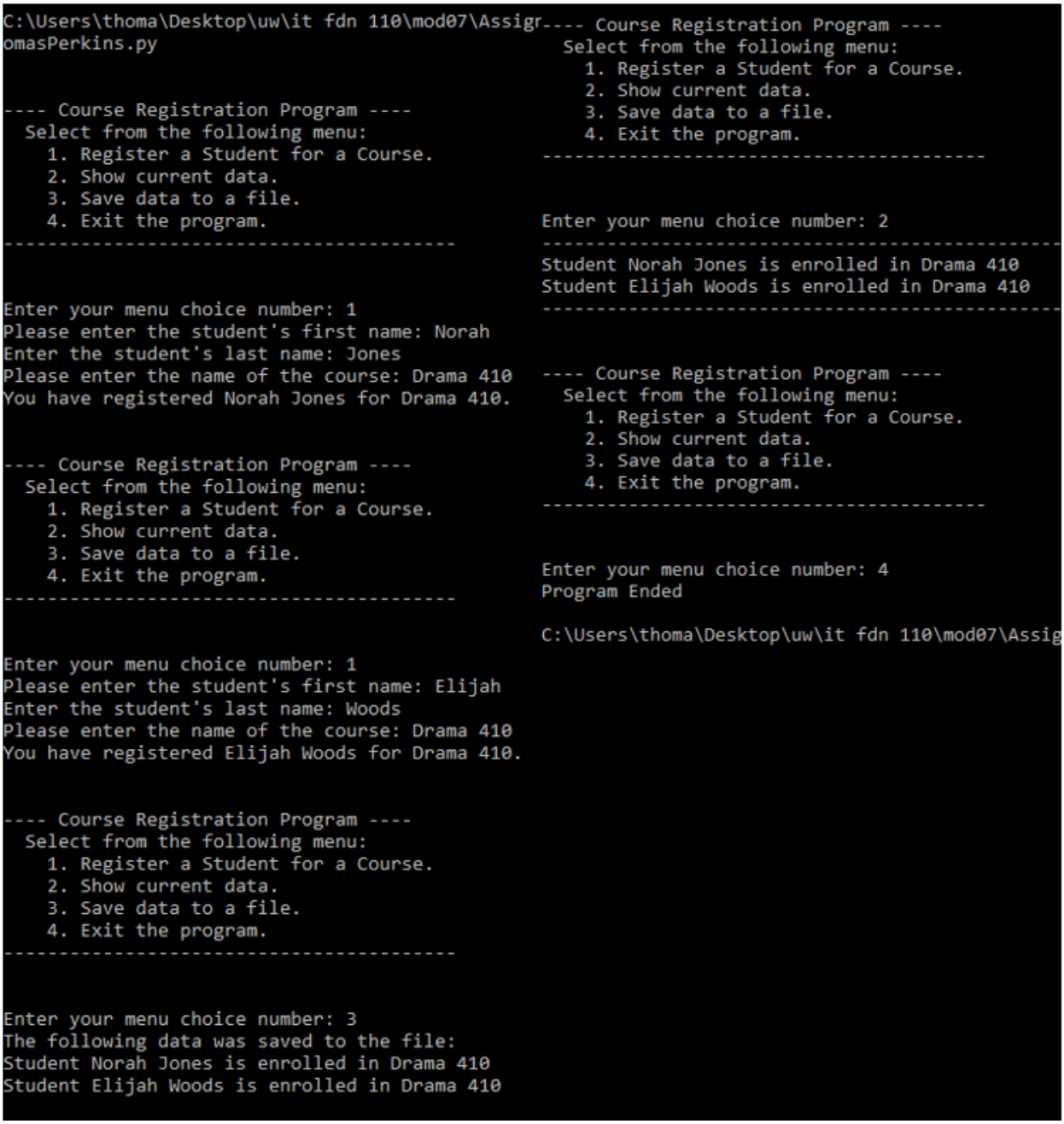

Figure 4: Windows Shell Output

Assignment07_ThomasPerkins.py | Enrollments.json | __in

```
[
    {
        "FirstName": "Norah",
        "LastName": "Jones",
        "CourseName": "Drama 410"
    },
    {
        "FirstName": "Elijah",
        "LastName": "Woods",
        "CourseName": "Drama 410"
    },
    {
        "FirstName": "Thomas",
        "LastName": "Perkins",
        "CourseName": "Python 101"
    },
    {
        "FirstName": "Vic",
        "LastName": "Vu",
        "CourseName": "Python 101"
    }
]
```

1 › CourseName

Figure 5: JSON File Output

```
Enter your menu choice number: 1
Please enter the student's first name: Thomas
Enter the student's last name: Perkins
Please enter the name of the course: Python 101
Inappropriate argument value (of correct type).

-- Technical Error Message --
Input is duplicate to an existing record.
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- 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.
----------------------------------------


Enter your menu choice number:
```

Figure 6: Error Handling for Duplicate Record

5.0 Summary

This Module was the first one to really deal with the Object-Oriented aspects of Python for the course, and it explains many of the behaviors and the syntax of the language that I have observed during the course. Although in the case of the assignment code using objects to form the list rather than strings offers only marginal benefit, it is obvious the clear advantages of class objects vs. simpler variables would bring with more complex data structures. I look forward to seeing where the course goes next, and to using these methods for future programming assignments.