

John Lewandowski

08/12/24

Foundations of Python

Assignment 07

<https://github.com/lewandj1/IntroToProg-Python-Mod07.git>

# Module 07- Classes and Objects

## Introduction

This report documents some of the features I learned to use within Python regarding the seventh assignment in the Foundations of Python course at the University of Washington. Key features included were as follows: Constructors, Attributes, Property, Inheritance, and Overridden Method. In each section I will briefly explain what the feature is and how I implemented the feature.

## Constructors

In this case I used constructors to initialize my object. By doing this whenever an instance of my object is called, the constructor would then be used, thereby initializing the attributes of the object. For the assignment two constructors were used for the Classes: Person and Student, shown in **Figure 1**.

```
class Person:
    """Defining that person has first and last name
    JWL, 08/11/24, Creation of Class"""

    def __init__(self, first_name: str, last_name: str):
        self.first_name = first_name
        self.last_name = last_name


class Student(Person):
    """Pulling information from Person to be included for the Student
    JWL, 08/11/24, Creation of Class
    """

    # call to the Person constructor and pass it the first_name and last_name data
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        # add an assignment to the course_name property using the course_name parameter
        self._course_name = course_name
```

**Figure 1: Example of the two Constructors being used.**

In the first case of Person, I set the two attributes of First Name and Last Name. Then for the Student, it would be First Name, Last Name, and Course Name. Course name being inherited from the Person, demonstrated later in the Inheritance section. By using the “Constructors we create, initialize, and return a new object” (Python, R 2023).

## Attributes

Attributes are the specific traits that belong to a class. In this assignment the Attributes are: First Name, Last Name, and Course Name. By splitting our class into Attributes we can then compile the information into constants because the Class attributes are shared by all the instances in the class (PythonTutorial, 2021). By doing that we allow our First Name, Last Name, and Course Name once pulled from the JSON file or input from our IO, to be recognized by our other Classes. Now we can utilize the class callout for other functions within the Class minimizing the excess code we need to have. This can be seen in **Figure 2**.

```
for student in student_data:
    file_data.append({'first_name': student.first_name,
                     'last_name': student.last_name,
                     'course_name': student.course_name})
file = None
```

**Figure 2: Example of Calling out the Attributes.**

As you can see, the image shows later in the code that we can individually call out our class: Student and pull the attributes: First Name, Last Name, and Course Name. By compiling it via a ‘for’ loop it allowed each student to be added to our dictionary list to then be written to the file as shown in **Figure 3**.

```

for student in student_data:
    file_data.append({'first_name': student.first_name,
                     'last_name': student.last_name,
                     'course_name': student.course_name})
file = None
try:
    file = open(file_name, "w")
    json.dump(file_data, file)
    file.close()
    IO.output_student_and_course_names(student_data=student_data)
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 is not None and not file.closed:
        file.close()

```

**Figure 3: Example of file\_data using the Attributes and being dumped into the file by JSON.dump function.**

## Property

When we need to call certain attributes but avoid calling them directly, Property's come into play. These allow the code to call certain attributes without actually modifying them. In doing so, we can have other instances of our code, that are outside the class, call our determined property value (GeeksforGeeks, 2020). By doing this we can control the format and value of the instance being called. As shown in **Figure 4**, you can see that for course name I decided to have the course name be put into a 'title' format whenever it is called. In doing so we assigned a 'property' to the attribute being called.

```
4 usages
@property # getter for course_name
def course_name(self) -> str:
    """
    ~~~~~
    Returns the course name
    :return: Course name
    """
    ~~~~~
    return self._course_name.title()
```

Figure 4: Example of Property for course name.

## Inheritance

Inheritance is when we want to define a class that inherits the methods and properties of another class (W3Schools.com). For this assignment I wanted to take the attributes: First/Last name and carry them onto the Student from the Person class. Thereby, allowing the code to reuse the First/Last name properties within my newly defined Student class. In other words, my Student class doesn't need to create the First/Last name properties because the code realizes that they are the same variables used in the Student class.

```

# call to the Person constructor and pass it the first_name and last_name data
def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
    super().__init__(first_name=first_name, last_name=last_name)
    # add an assignment to the course_name property using the course_name parameter
    self._course_name = course_name

4 usages
@property # getter for course_name
def course_name(self) -> str:
    """
    ~~~~~
    Returns the course name
    :return: Course name
    """
    ~~~~~
    return self._course_name.title()

@course_name.setter # setter for course_name
def course_name(self, value: str) -> None:
    if value.isalnum():
        self._course_name = value
    else:
        raise ValueError("Course name must be alphanumeric.")

# Override the __str__() method to return the Student data
def __str__(self) -> str:
    """
    ~~~~~
    String call for the course_name
    return: String for Student as a csv file
    """
    ~~~~~
    return f"{super().__str__()}, {self.course_name}"

```

**Figure 5: Inheritance of First/Last name to the Student from Person.**

Now the Student classes will inherit the properties: First name and Last name. The `super()` function is superimposing those two properties/attributes into the definition and thereby pushes those two attributes into the `str` when calling the 'Student', lastly we add the course name as our final attribute for

Student.

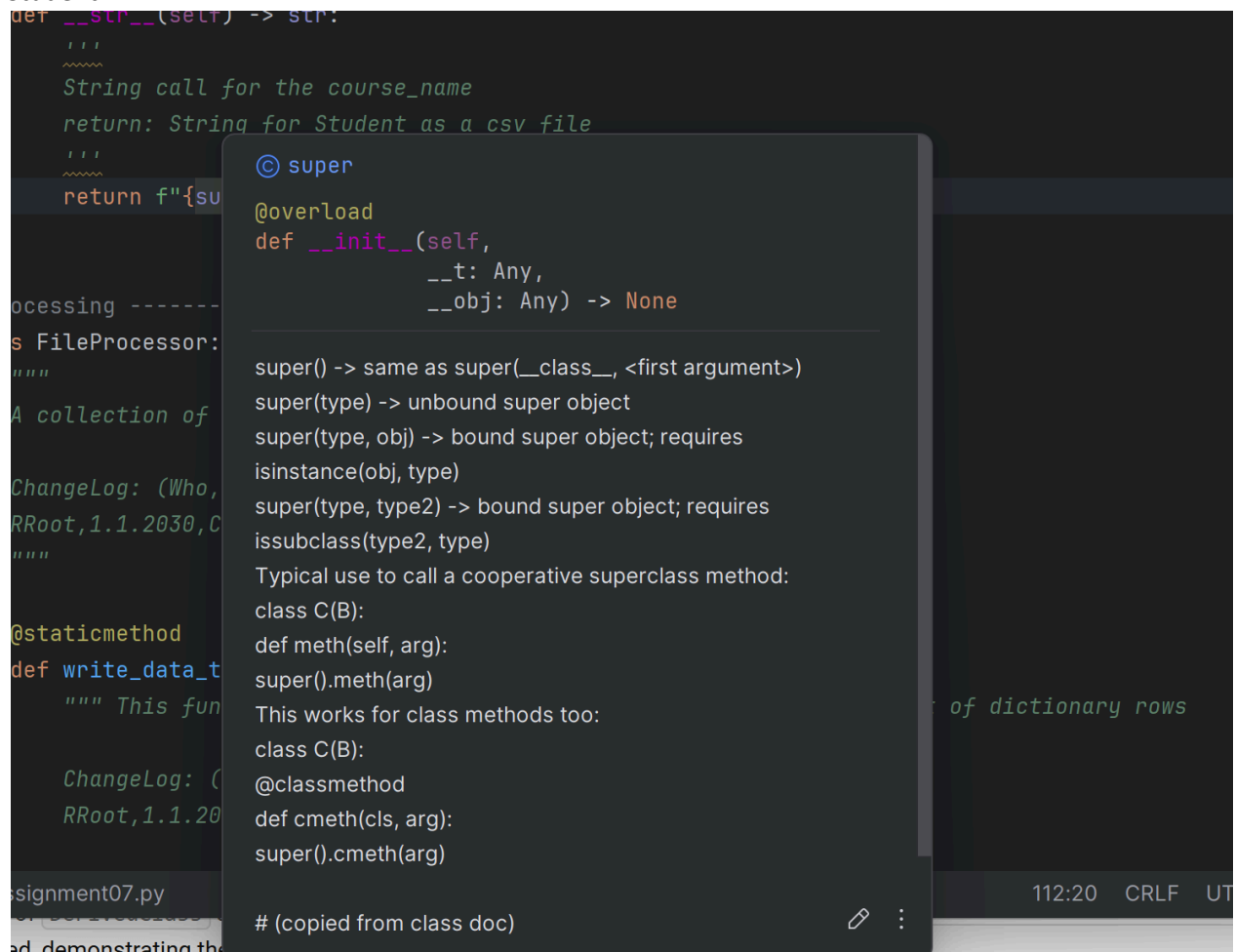
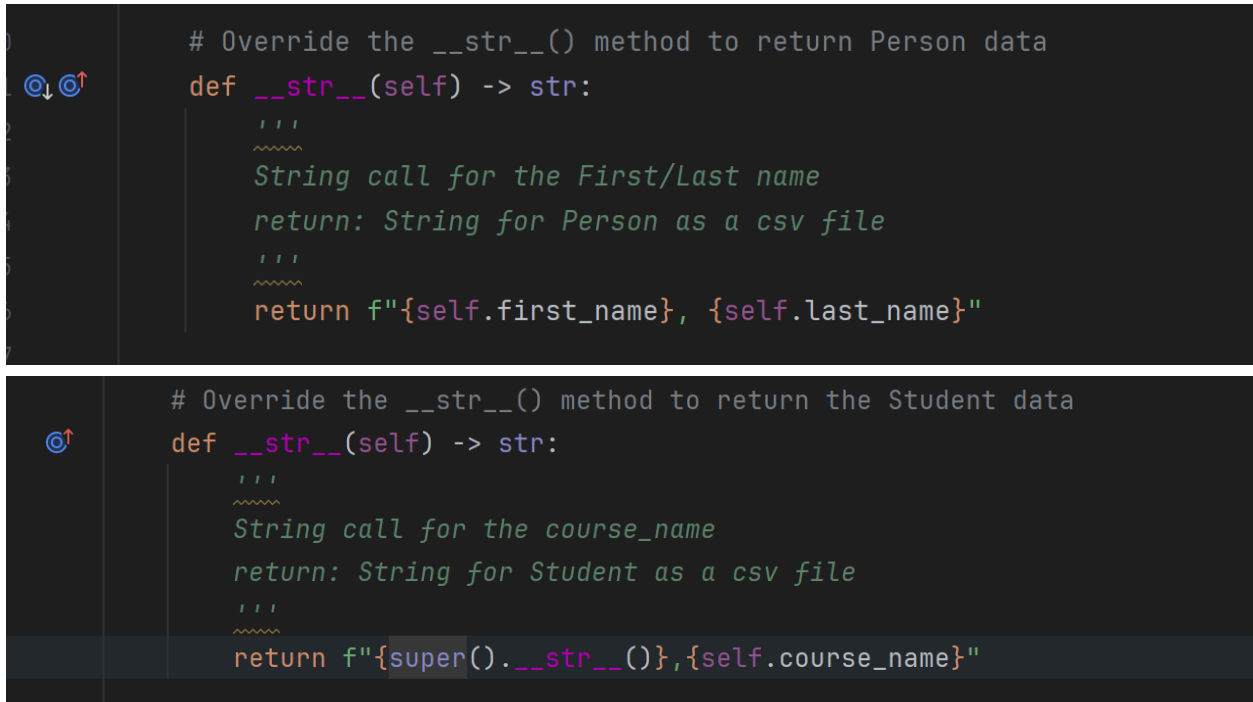


Figure 6: Pycharm definition for the `super()` function.

## Overridden Method

As seen in **Figure 6** the override function is when we want to push certain attributes into our data. For the case of this assignment it was the First Name and Last Name being added to the attributes of 'Student'. "Using the override method allows specific implementation of a method that is already provided by one of its super-classes or parent classes" (GeeksforGeeks, 2024c). In this case we are overriding the string for our 'Student' which we arbitrarily made blank for the purposes of formatting the key's, shown in **Figure 7**.



```
# Override the __str__() method to return Person data
def __str__(self) -> str:
    """
    String call for the First/Last name
    return: String for Person as a csv file
    """
    return f"{self.first_name}, {self.last_name}"

# Override the __str__() method to return the Student data
def __str__(self) -> str:
    """
    String call for the course_name
    return: String for Student as a csv file
    """
    return f"{super().__str__()}, {self.course_name}"
```

**Figure 7: Setting the First/Last attributes for Student(Person) class. Then we override the string by imposing our super() to input the First, Last, and then course name.**

As seen above, the override method basically just calls the original string of 'first\_name and last\_name' and then adds to it the 'course\_name' in doing so, adding the course name attribute to our Student.

## Summary

Following the Module 07 videos, reading/watching the additional videos and modules in tasks 2 and 3 of the assignment file and referencing helpful websites on Python usage, I was able to successfully complete the assignment. I learned how to create constructors and then implement them so that attributes can be shared between classes. Then I learned how to take attributes and call them within my functions, in a different way than calling the variable directly. Defining Properties allowed me to inherently set the method of how my code called certain attributes, allowing me to formulate the data into lists that could be written to the file. This was important because by doing so, I didn't have to compromise the data, instead I was able to call the local values. After properties, I learned how to inherit certain attributes from a class to another class, thereby minimizing the amount of code being duplicated when duplicate information types is being used. Finally, I learned how to override methods allowing me to superimpose data into my list by taking the inherited attributes and pushing them as strings in my defined constructor, and thereby keeping my data formatted with accessible keys/functions. These skills greatly increase my ability to develop as a coder and progress through my journey as more advanced techniques are brought to my attention via the next assignment.

## References

- Python, R. (2023, September 25). Python Class Constructors: Control your object instantiation. <https://realpython.com/python-class-constructor/>
- Python Tutorial. (2021, October 17). Understanding Python encapsulation clearly by practical examples. Python Tutorial - Master Python Programming for Beginners From Scratch. <https://www.pythontutorial.net/python-oop/python-private-attributes/>
- GeeksforGeeks. (2020, November 23). Accessing attributes and methods in Python. GeeksforGeeks. <https://www.geeksforgeeks.org/accessing-attributes-methods-python/>
- W3Schools.com. (n.d.-c). [https://www.w3schools.com/python/python\\_inheritance.asp](https://www.w3schools.com/python/python_inheritance.asp)
- GeeksforGeeks. (2024c, August 7). Method overriding in Python. GeeksforGeeks. <https://www.geeksforgeeks.org/method-overriding-in-python/>
-