James Crockett
2021-Nov-29
Foundations of Programming (Python)
Assignment 08

# Object Oriented Programming

## Introduction

Python is an Object-Oriented Programming language, OOP. Everything in the language can be considered an object on a conceptual level according to the week 8 PDF[1]. The examples provided from the PDF go on to show how classes are a foundation in OOP by providing a structure for the rest of the program. Each Class that is created can be constructed to perform specific functions that other objects in the program can inherit.

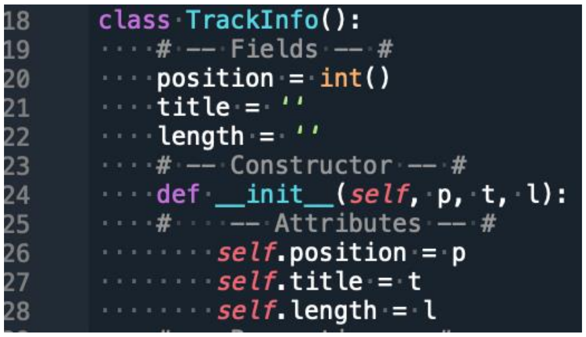
## Initial Approach

I couldn't start Assignment 08 at the beginning of this week because I didn't have a solid grasp of OOP since it is such a new concept to me. The OOP concepts have quite a few details that work together (Class object, Fields, Constructor, Attributes, Properties, Methods). I worked through the first few labs successfully until I encountered errors on Lab08D with calling the objects that use getter and setter methods.

## Understanding OOP Classes

Classes are the blueprint of custom objects and are composed of: Fields, Constructor, Attributes, Properties, and Methods.

Fields are where data is stored like a variable. Fig. 1, line 24, a constructor is defined with an __init__ method and is what allows an object to be called in a similar way to a function. Lab08-B provided practice on how to create the constructor a then call the attributes as shown below. Each attribute method is preceded by the naming convention label, self. Self is used to identified an attribute that used to instantiate the object.



```
18  class TrackInfo():
19      # -- Fields -- #
20      position = int()
21      title = ''
22      length = ''
23      # -- Constructor -- #
24      def __init__(self, p, t, l):
25      #    -- Attributes -- #
26          self.position = p
27          self.title = t
28          self.length = l
```

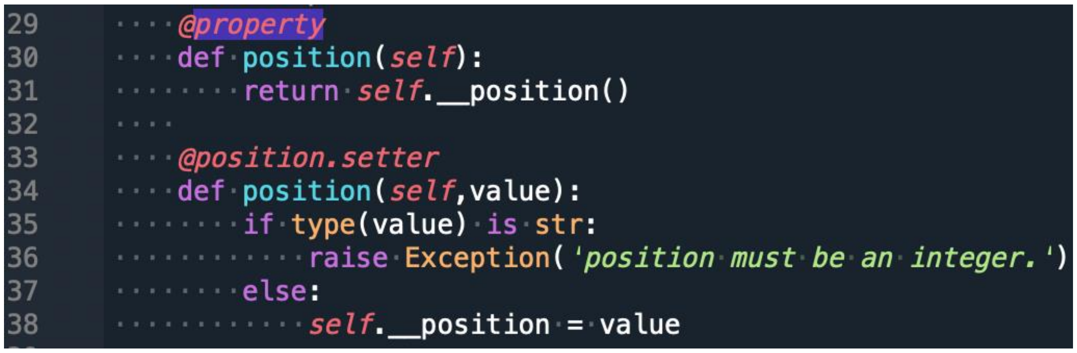
*Figure 1 - Constructors and Attributes*

[1] Dirk Biesinger, FDN_Py_Module_08.pdf, 1.

## Property Methods: Getters and Setters

It’s considered a best practice to always use a property to access data in a class. Properties control access to attribute values, they act as gatekeepers or methods. Each attribute has two kinds of property methods: getter (for reading), and a setting (for writing) to the attribute. With Python, a double underscore must be prepended to the attribute name to make it private.

The proper way to set up getters and setters is to define the getter first followed by the setter property.

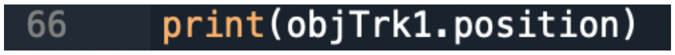


```
29      @property
30      def position(self):
31          return self.__position()
32      
33      @position.setter
34      def position(self,value):
35          if type(value) is str:
36              raise Exception('position must be an integer.')
37          else:
38              self.__position = value
```

*Figure 2 - Lab08D: Initial implementation of getter and setter property methods, line 31 shouldn’t have trailing parentheses.*

The biggest difference in my object class for Lab08D from previous labs is now my Object had getter and setter methods defined as shown in Figure 2 above.

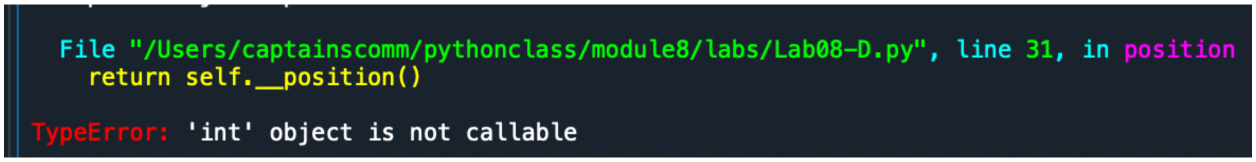
In Figure 3, I continued calling the attribute, .position, the same way I had called in in previous Lab08 labs leading up to Lab08D.



```
print(objTrk1.position)
```

*Figure 3 - Calling Object method .position*

Everywhere I called one of the three property methods, (.position, .title, or .length), the Spyder console output the “object is not callable” error as shown in Figure 4 relative to the property method I was trying to call.



```
  File "/Users/captainscomm/pythonclass/module8/labs/Lab08-D.py", line 31, in position
    return self.__position()

TypeError: 'int' object is not callable
```

*Figure 4 – ‘object is not callable’ from calling object method .position.*

I went back to Line 31, Figure 2, to try to understand how I was triggering this error. At this point, I started digging into Chapter 8 of the book[2] and the book source code files where it describes accessing Attributes. I noticed the syntax from the source code files from the book for the property return statements did not contain the trailing parentheses I used in Line 31, Figure 2. Once I removed the

[2] Michael Dawson, Python Programming for the absolute beginner, 3rd Edition, 238-240.

trailing parentheses from all property return statements, the Spyder errors went away. Syntax issues were the problem here, but it took time to research and understand that as the root cause.

## Instance and Static Methods

Methods help organize blocks of code to perform a particular job. The first attribute supplied to a method is the "self" reference as noted in the Module 8 PDF.[3]

There also static methods that can be used to keep track of data affecting the object at a class level instead of an instance level. Class "fields" and "static methods" are commonly used to keep track of information about the class.

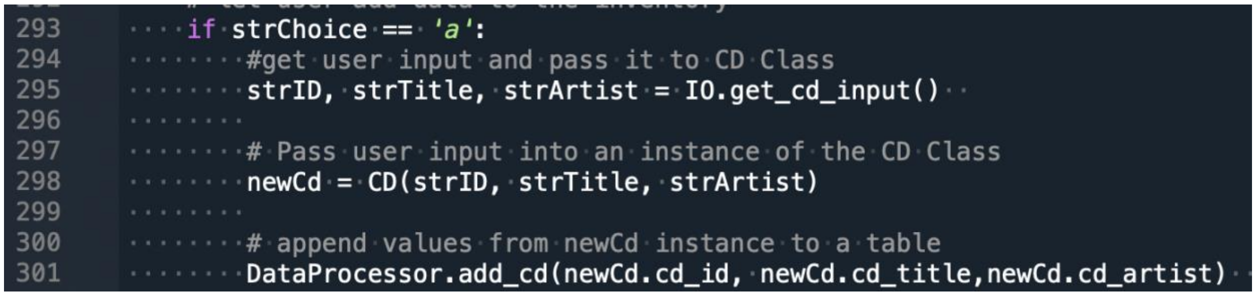
Static methods don't use the "self" attribute because self is used for instance level code blocks. Self is only used when passing a reference of the object in.

Key takeaways for methods:

- Classes that focus on **processing data** focus on using static methods.
- Classes that focus on **storing data**, instance methods are used that require the "self" attribute.

## Pulling Attribute values from Object Instance

I was receiving errors in Spyder when I attempted to pass the reference of the Object into the add_cd function to be appended to a table. I performed a print(newCd) and discovered Spyder output the memory location to the object and not the values in the attribute instances of the object for ID, Title, and Artist. I reviewed my previous labs to review what syntax I had used to get data from object attributes and updated line 301. Figure 15, line 302 includes a list of the object name and attribute names that needed to be appended. The correct values began to display in the table output and Spyder no longer was throwing errors for this block of code.



```
293     if strChoice == 'a':
294         #get user input and pass it to CD Class
295         strID, strTitle, strArtist = IO.get_cd_input()
296         
297         # Pass user input into an instance of the CD Class
298         newCd = CD(strID, strTitle, strArtist)
299         
300         # append values from newCd instance to a table
301         DataProcessor.add_cd(newCd.cd_id, newCd.cd_title,newCd.cd_artist)
```

*Figure 5 – My solution to the syntax challenge with pulling data from CD object.*

[3] Dirk Biesinger, FDN_Py_Module_08.pdf, 10.

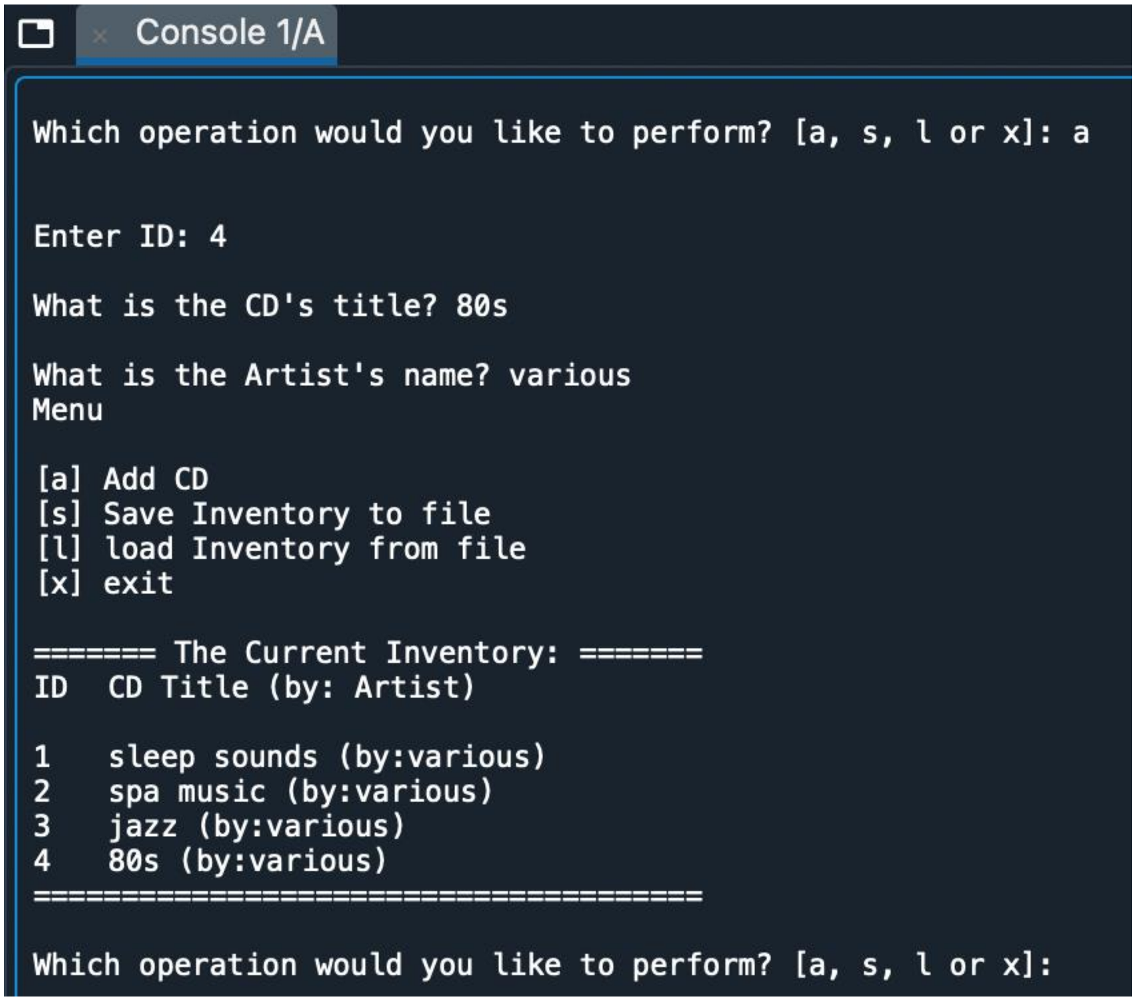
## Summary

I think one of my biggest challenges is that we are covering so much material in the last few weeks. The pace of this 10-week course requires quite a bit of personal immersion into new subject matter being presented, while trying to retain concepts from previous weeks.  I have learned a lot in a short period of time, but I still feel like I need to work with the concepts more to really ingrain them into my practical application of what is being conveyed to become a more effective programming student.  I feel as though I know just barely enough to implement basic concepts, but not at the level where I want to be. I want to have a better command of programming and Python syntax and I think I just need to put in more coding iterations with what I've learned as I take on new projects.

## Appendix

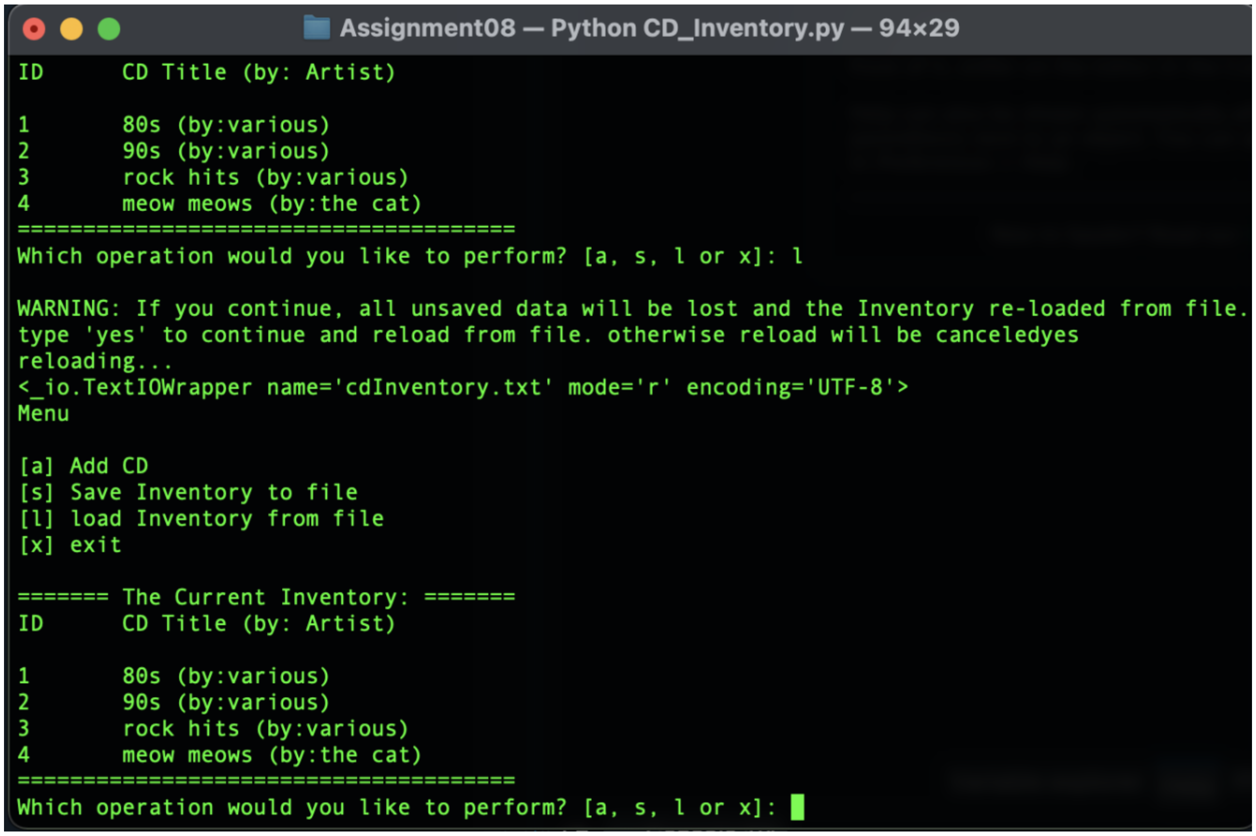
The following pages in this appendix show screen captures of:

- Figure 6, Spyder Console code execution.
- Figure 7, Mac Terminal code execution.
- Github, Assignment 8 Repository: https://github.com/jamescisonline/Assignment_08



```
Which operation would you like to perform? [a, s, l or x]: a


Enter ID: 4

What is the CD's title? 80s

What is the Artist's name? various
Menu

[a] Add CD
[s] Save Inventory to file
[l] load Inventory from file
[x] exit

======= The Current Inventory: =======
ID  CD Title (by: Artist)

1   sleep sounds (by:various)
2   spa music (by:various)
3   jazz (by:various)
4   80s (by:various)
======================================

Which operation would you like to perform? [a, s, l or x]:
```

*Figure 6- Spyder console runtime.*



```
ID      CD Title (by: Artist)

1       80s (by:various)
2       90s (by:various)
3       rock hits (by:various)
4       meow meows (by:the cat)
======================================
Which operation would you like to perform? [a, s, l or x]: l

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.
type 'yes' to continue and reload from file. otherwise reload will be canceledyes
reloading...
<_io.TextIOWrapper name='cdInventory.txt' mode='r' encoding='UTF-8'>
Menu

[a] Add CD
[s] Save Inventory to file
[l] load Inventory from file
[x] exit

======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       80s (by:various)
2       90s (by:various)
3       rock hits (by:various)
4       meow meows (by:the cat)
======================================
Which operation would you like to perform? [a, s, l or x]: 
```

*Figure 7 - Mac Terminal runtime.*