Nabil Knight

2022-Mar-20

IT FDN Programming: Python

Assignment 08

Object Oriented Programming: Using Classes, Writing and Instantiating Objects

# Introduction:

This week in the Foundation of Programming course we began diving into Object Oriented Programming. Some of the topics we went over were working with classes in more detail and how a class is essentially a blueprint of an object. We also went over methods which are essentially the same thing as a function, they allow you to organize your statements into logical components that can be called using the methods name. And we also dove into constructors, these allow a program to ensure proper data types in their fields. Additionally, we dove into properties, the self keyword and attributes. We put our learnings into practice by taking these concepts and building out the CD Inventory program and applying classes, methods, and constructors to get the program to work properly. Please find my location to my GitHub repository [here](https://github.com/nabilk81/Assignment_08).

# Classes:

This week we began working with classes in more depth. As stated, a class is like a blueprint for an object, when you use that object, you are instantiating or making an instance of that class. Dirk gave the example of having a blueprint of a house and then each owner can customize that house in their own way. It is the same blueprint across each house but every instance of it has its own layout and design. In object-oriented programming, functions are typically called methods and constants / variables are generally called fields. Example code of fields below:

**class** **TrackInfo**():

# --- Fields --- #

position = ''

title = ''

length = ''

# Constructors, Self, Attritubes, Properties:

Constructors are an efficient way of ensuring the proper data types are being used in the fields. One such example is the dunder init method or: \_\_init\_\_(). Example code below:

# --- Constructor --- #

**def** **\_\_init\_\_**(self, pos, titl, leng):

self.position = pos

self.title = titl

self.length = leng

Python will implicitly call the dunder init method and pass any argument that is provided, and in the example above, we see: pos, titl, and leng being passed as arguments. The self keyword as Dirk mentions is not an official pythonic keyword but is widely used and other programmers will be expecting it. The self keyword is the first parameter in every method, in a similar way you can pass the reference to a list around, a programmer can pass the reference to the object around the program. This means we can call a method elsewhere in the program and pass the reference of the object to that new piece of code. Something, of note, is that if you forget to pass self in the object you will receive an error for this. Attributes are fields that hold data and you have little to no control over what gets set into them unless you write specific code to control this. Example code for attributes is below:

**class** **CanOnAString**():

fldMessage = ''

**def** **\_\_init\_\_**(self, msg):

self.message = msg

objCan1 = CanOnAString('There is no spoon')

print('1st: {}'.format(objCan1.message))

print('2nd: {}'.format(objCan1.fldMessage))

As you can see above, fldMessage gets set as a field or attribute for the class CanOnAString, however because there was no message it was set with in the dunder init method, the outout will not return a value.

Properties are a way to control the values assigned to attributes in a programs class, this allows them to become private. Typically, you create two properties for each variable, one called the setter (mutator) and one called the getter (accessor). To make an attribute private in python you prepend a double underscore:\_\_propertyname. Generally, you defin the getter first and then the setter. It is a best practice to work with data in a class through your property, it creates a layer of abstraction or hiding the functionality from the user. LAB8-D has us work with properties in more detail:

**class** **TrackInfo**():

# --- Fields --- #

position = ''

title = ''

length = ''

# --- Constructor --- #

**def** **\_\_init\_\_**(self, pos, titl, leng):

self.\_\_position = pos

self.\_\_title = titl

self.\_\_length = leng

@property

**def** **position**(self):

**return** self.\_\_position

@property

**def** **title**(self):

**return** self.\_\_title

@property

**def** **length**(self):

**return** self.\_\_length

@position.setter

**def** **position**(self, value):

**if** int(value) **is** str(value):

**raise** Exception('please enter a number')

**else**:

self.\_\_position = value

@title.setter

**def** **title**(self, value):

**if** str(value).isNumeric():

**raise** Exception('please enter a Title')

**else**:

self.\_\_title = value

@length.setter

**def** **length**(self, value):

**if** str(value).isNumeric():

**raise** Exception('please enter a Title')

**else**:

self.\_\_length = value

# --- Methods --- #

cdinfo1 = TrackInfo(1, 'Song 1', '3:00')

cdinfo2 = TrackInfo(2, 'Song 2', '3:30')

cdinfo3 = TrackInfo(3, 'Song 3', '2:00')

print('Track 1: Position: {}, Title: {}, Song Length: {}'.format(cdinfo1.position, cdinfo1.title, cdinfo1.length))

print('Track 2: Position: {}, Title; {}, Song Length: {}'.format(cdinfo2.position, cdinfo2.title, cdinfo2.length))

print('Track 3: Position: {}, Title: {}, Song Length: {}'.format(cdinfo3.position, cdinfo3.title, cdinfo3.length))



Figure 1 - Lab8D Results

As you can see above, each of the attributes, position, title and length all get their own respective getter and setter. I then define three values using the TrackInfo class: cdinfo1, cdinfo2, and cdinfo3 and then print out the results in Figure 1.

# Methods, etc:

Methods similar to functions, they allow a programmer to organize their code into different segments and the main difference is that a method also submits a reference to the object. Generally, the first attribute in a method is the self reference, (self, argument) The string method or \_\_str\_\_ returns some or all of the objects data in a string. Python implicitly inherits the method from the class object if one is not set. You can define your own method if you need to use it for something more suitable to your needs. In Lab8E we were tasked to copy the LAB8D and add the \_\_str\_\_method to format our attributes, code and results below:

**def** **\_\_str\_\_**(self):

result = '{}, {}, ({})'.format(self.\_\_position, self.\_\_title, self.\_\_length)

**return** result

cdinfo1 = TrackInfo(1, 'Song 1', '3:00')

cdinfo2 = TrackInfo(2, 'Song 2', '3:30')

cdinfo3 = TrackInfo(3, 'Song 3', '2:00')

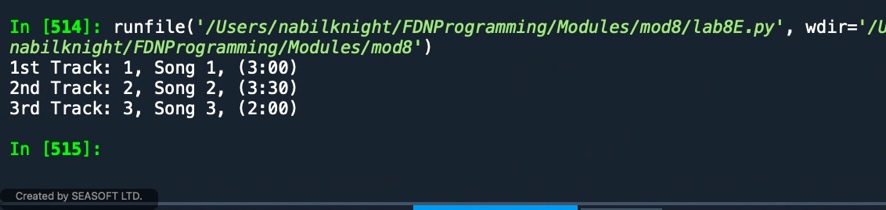


Figure 2 - LAB8E Results

As you can see above the addition of the string method formatted the data for each track listing as intended without needing the same amount of code.

Static methods are used when you want to have a method to be called on a class level and not on an instance level. Using @staticmethod issued to indicate this method type and you don’t need to pass in a self-attribute because there isn’t a reference to the calling object. A general rule of thumb is to use a static method when you need to process data and when you need to store data it is best to instantiate or use an instance method. Private methods indicated by two leading underscores as seen in the code for LAB8E in each of the getter and setter properties and is used to maintain the fields so they aren’t changed or overwritten.

# Assignment:

I had a real doozy of a time on this week’s assignment, I think it was partly from the new concepts of OOP and the maybe the combination of all of the past learnings to date. But I couldn’t for the life of me get the inputted data into memory to form a list. I was not using the dunder string method correctly and most likely not accessing it correctly. Oddly, I felt like I was following along pretty well in the labs but when the assignment came around clearly there was a gap in my knowledge. To show some of the new code that was used in the week’s assignment, in the CD class I included the getters and setters for each of the data points that were needed for our CD Inventory. Please see below:

**class** **CD**:

"""Stores data about a CD:

properties:

cd\_id: (int) with CD ID

cd\_title: (string) with the title of the CD

cd\_artist: (string) with the artist of the CD

methods:

"""

**def** **\_\_init\_\_**(self, ID, title, artist):

self.\_\_cd\_id = ID

self.\_\_cd\_title = title

self.\_\_cd\_artist = artist

@property

**def** **cd\_id**(self):

**return** self.\_\_cd\_id

@cd\_id.setter

**def** **cd\_id**(self, value):

**if** type(value) == int:

self.cd\_id = value

**else**:

**raise** Exception('cd\_id needs to be an integer')

@property

**def** **cd\_title**(self):

**return** self.\_\_cd\_title

@cd\_title.setter

**def** **cd\_title**(self, value):

**if** type(value) == str:

self.cd\_title = value

**else**:

**raise** Exception('cd\_id needs to be a CD Title in string format')

@property

**def** **cd\_artist**(self):

**return** self.\_\_cd\_title

@cd\_artist.setter

**def** **cd\_artist**(self, value):

**if** type(value) == str:

self.cd\_artist = value

**else**:

**raise** Exception('cd\_id needs to be a CD Artist in string format')

**def** **\_\_str\_\_**(self):

lst\_Inventory = '{},{},{}'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)

**return** lstOfCDObjects.append(lst\_Inventory)

I am not sure I set the dunder string method correctly and thus is likely part of my issue in being able to save the data into memory. My perception and is likely wront, was that I felt like I was just using code from last weeks assignment to augment the first getter and setter portion and the part I really was confused with was how to integrate them both. It would be nice if we could walk through this on Wednesday. Figures below showing the issues I am experiencing (pretty sure I am not saving anything and thus not able to load anything back into memory):

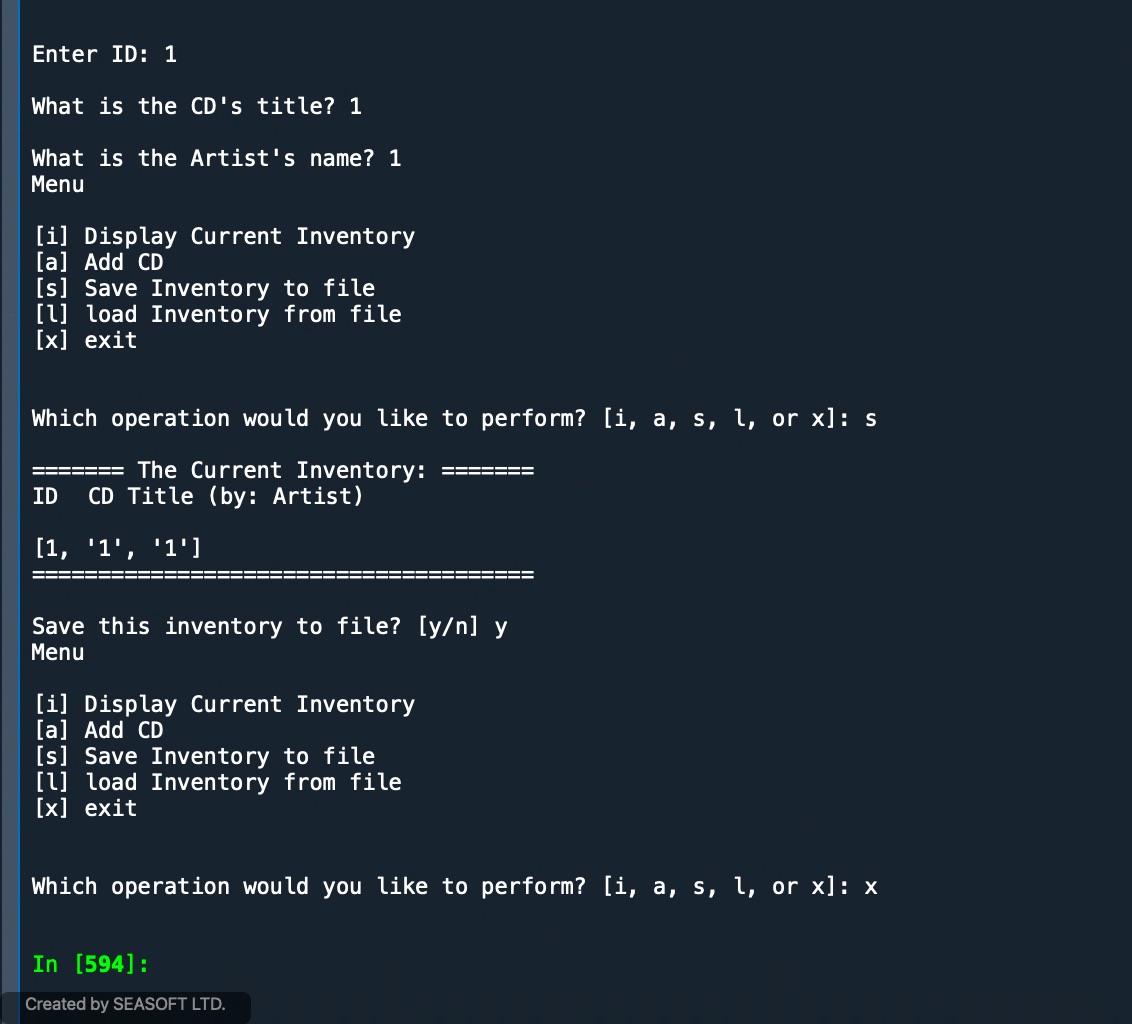


Figure - Results from Assignment

# Conclusion:

During the week we began learning about classes, attributes, methods, instantiating object and whole host of topics that relate to object oriented programming. To say it was confusing wouldn’t do this justice. I had a tough time wrapping my head about this and trying to be able to utilize all weve learned to date but then use it a different way. To put our learnings into practice we used our CD Inventory framework to use the same menu style program but using object oriented programming. The labs in their silos all made sense but when I would try and take a step back to look at the big picture I had a tough time connecting the dots between concepts. As I was doing the assignment, I felt like I kept tripping over myself as I tried to get the program to work and wasn’t able to in the end. It would nice if we could walk through this on Wednesday, so that I could this in working order and learn from my mistakes.