< Bryn Dunbar >

< 2020-March-16 >

< IT FDN 100: Foundations of Programming (Python) >

< Assignment 08 >

Complete TODO’s of Assignment 08 Starter Code

# Introduction

Module 8 introduced the concept of object-oriented programming. Programming with objects is more efficient because there is less memory and storage required because when classes are defined, only the description of the object is required. Object-oriented programming also makes things simple because each entity is defined as a separate object and code can be re-used and repeated for each different object. Classes are the blueprints for objects. You can write one class and use it for ten different objects.

# Topic 1 – Classes are Blueprints

Classes are defined with fields, constructors to initialize objects, attributes, properties and methods. Data for a class is stored in fields. Constructors are dedicated methods that are created automatically as an object is created. Constructors are used for instantiating objects. The **\_\_init\_\_()** method is what is used as the constructor. Constructors always have at least one argument known as **self**, which refers to the instance of the object. Constructors can also have additional arguments, if desired. Python is unique in the fact that it uses both attributes and fields for objects. Most programming languages do not use both. Classes have attributes and objects also have attributes. The constructor not only instantiates an object, but it is also used to ensure that the object has any desired or required attributes. Attributes for objects are characteristics of an object and attributes of a class are characteristics of a class. Properties are special methods used to interact with attributes. A property is created to set values for attributes and a property is created to access those values. The properties used to set values are called setters or mutators. The properties used to access or get values are called getters or accessors. In summation, attributes are set and accessed by properties. You can make a property private by including a double underscore or a dunder before the attribute name. A private property is designed to be only accessible from within a class. When a property is private, it is advised to not access it directly, even though it is possible. Methods are similar to function. They describe an action or tell the program to do something. The difference between a method and a function is that a method refers to an object and a function does not have a reference to an object.

# Topic 2 – Applying what I’ve Learned

In Assignment 08, the main task is to use the code that we have previously been working with for a CD Inventory, but to build and use objects within the code. To begin the assignment, I copied and pasted code from my Assignment 07 into the appropriate TODO’s in the starter code for Assignment 08. The starter code was helpful because in the DATA section, a list of objects known as **lstOfCDObjects**, was already defined. I replaced instances of **lstTbl** (the list of lists from Assignment 07) with a list of objects (**lstOfCDObjects**).



Figure 1 – Example of Code from Assignment 07 Needing Adjustments for Assignment 08



Figure 2 – Example of Adjusted Code for Assignment 08, using the suggested Starter Code names

I also used Assignment 08 Starter Codes class names and function names, so I had to change **FileProcessor** to **FileIO**, and **read\_file** to **load\_inventory**.

After I replaced necessary components of my recycled code from Assignment 07, to fit the starter code of Assignment 08 and to use a list of objects, I tackled the task of defining objects, object attributes and properties in the class CD. This portion of the code is new because it defines objects, which were previously just lists. To begin this duty, I defined an object for CD by using the constructor **\_\_init\_\_()** method. I added the unofficial keyword **self** as an argument for the constructor to refer to the object. I also added **cd\_id**, **cd\_title**, and **cd\_artist** as arguments for the object. Then I created attributes which are internal fields that hold data. Then I continued on by defining getter and setter properties for the attribute values in the class CD. The getter properties begin with the decorator **@** and the word **property**. I followed the requirements of defining a property with a getter method by using the **self** keyword and I made the properties private by pre-pending with a dunder or double-underscore. After each getter method, I also used a setter method for the property by using the **@** decorator, the attribute and **.setter**. The getter and setter methods are used to control values assigned to attributes. Getters are great for formatting fields and they are required for accessing private attributes and fields. Setter methods can be used for validating values and for structured error handling.

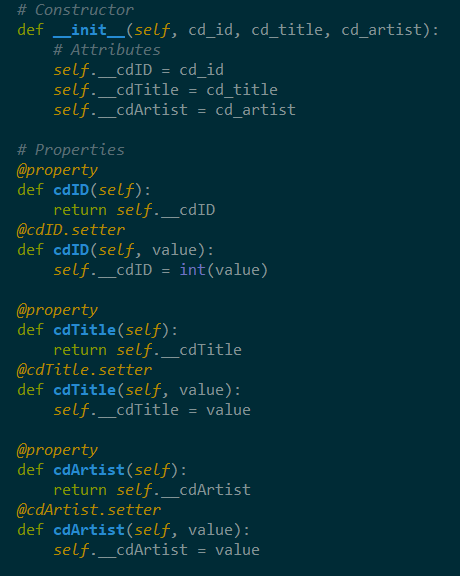


Figure 3 – Object Constructor and Getter and Setter Properties

Since Assignment 08 requires the implementation and usage of objects, there is no need to have a list of dictionaries anymore. The references to lists of dictionaries need to be replaced with lists of objects. I copied the pre-existing code used in Assignment 07 for much of Assignment 08. However, since we are no longer using dictionaries, I had to eliminate the references to dictionaries in my add\_cd method.



Figure 4 – Assignment 07 code Referring to a List of Dictionaries

I am thankful to the programming guidance from Douglas Klos. He lead me to use this line of code for my **add\_cd** function, to refer to the list of CD objects:



Figure 5 – Code From Douglas Klos to Refer to a List of Objects that I included in my Assignment 08

The next thing that I had to fix was my remove\_cd method. When I copied and pasted the code from my Assignment 07, it did not work anymore. After a closer look at the code, I realized that it was referring to a dictionary row named ‘ID’.

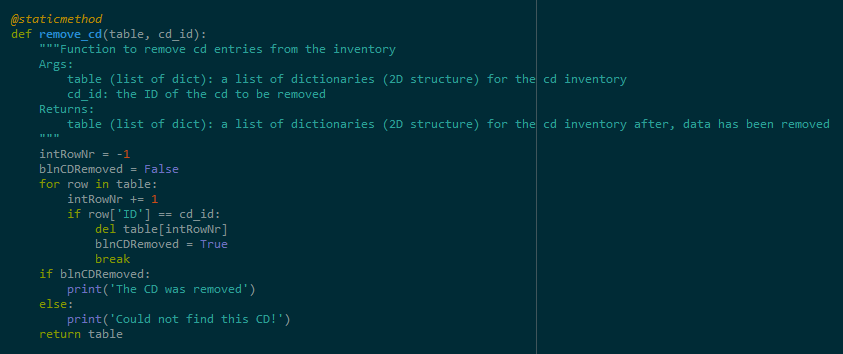


Figure 6 – Code From Assignment 07 that Doesn’t work in Assignment 08 because of its Reference to a Dictionary Row

To fix this, I rewrote a portion of the code to refer to the cdID row of my list of objects. After I modified the code, it now compares the user input entered for the cd ID (cdNum) with the cdID row and if it is the same ID, it will delete the cd with that ID number.

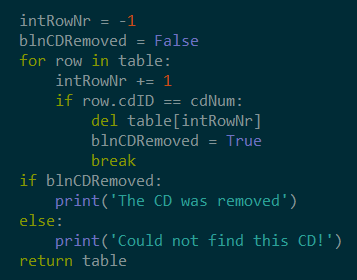


Figure 7 – Modified Code to refer to a row in the List of Objects for Assignment 08

I had a lot of miscellaneous bugs and errors in my code at first. I am again, thankful for the help that Douglas Klos provided me with. Some of the bugs that I had in my code were due to simply copying and pasting code from Assignment 07 into different parts of my Assignment 08 starter code and then reworking or tweaking parts attempting to make the script work. I also ended up getting my getter and setter methods mixed up with the values and the dunder (private) properties in the wrong places. Luckily, again, I got the help that I needed when I asked for it. I learned my lesson that refactoring code instead of simply copying and pasting is a better way to accomplish programming tasks.

# Summary

I definitely struggled with this Assignment because I ran into a lot of bugs that I didn’t know how to solve, without seeking help. I think that the general concept of object-oriented programming is clear and I can see how it adds efficiency and reusability. Naturally, it makes a lot of sense to create one class for many objects, versus one class for every object. However, I think trying to use the concepts and structures we have previously learned for building a cd inventory in python and then meshing those together with new objects was hard for me to comprehend. I am looking forward to hearing and seeing more examples and explanations in class tomorrow.

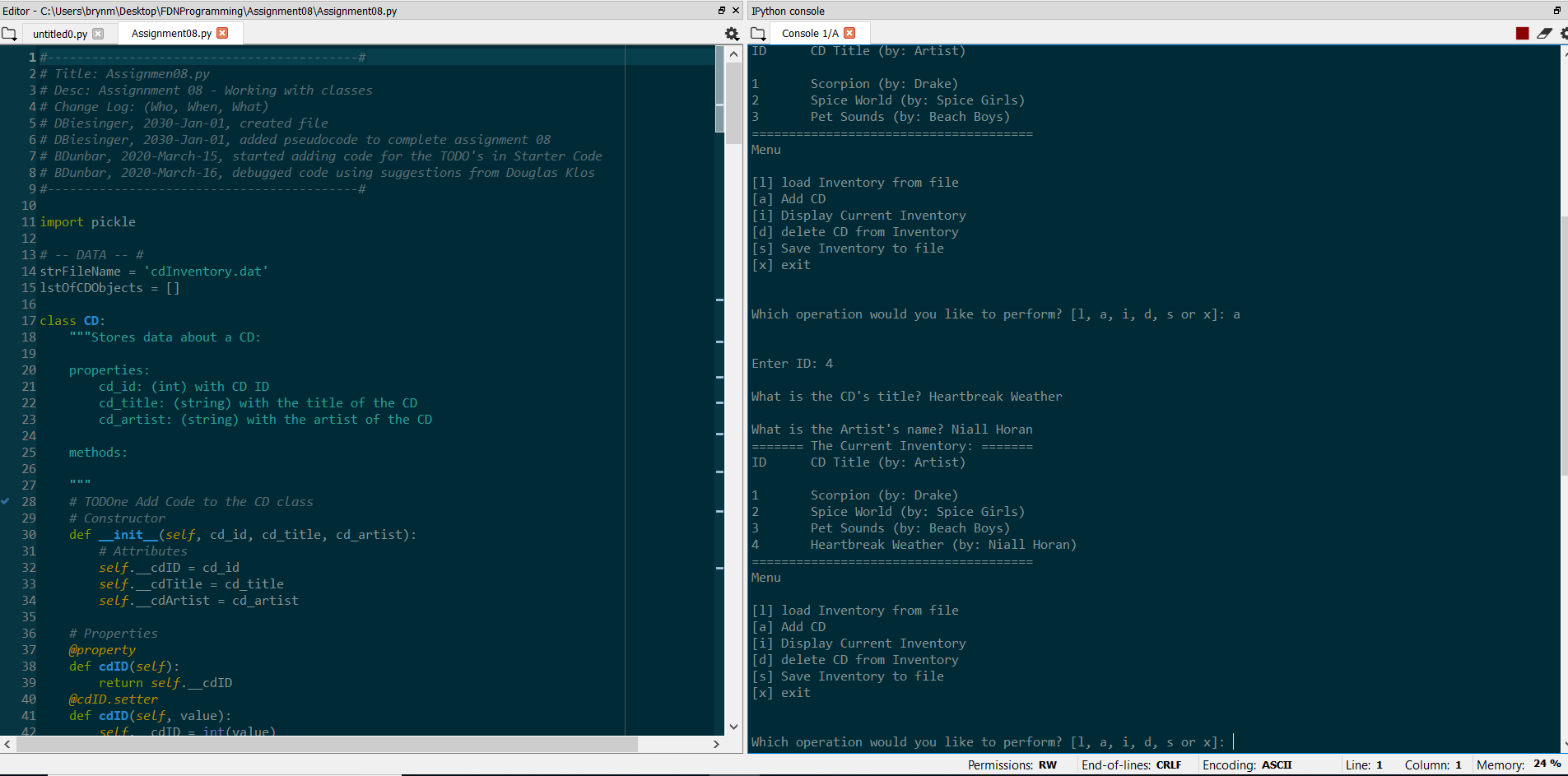


Figure 8 – Image of Working Script in Spyder – Add CD

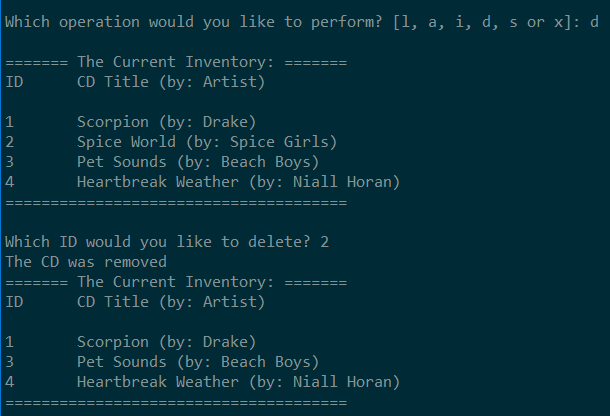


Figure 9 – Another Image of Working Script in Spyder – Remove Or Delete CD

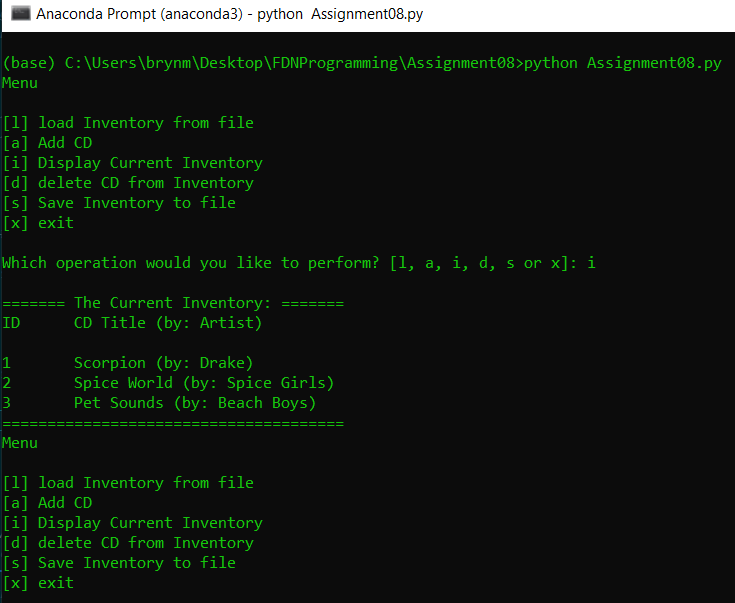


Figure 10 – Image of Working Script in Anaconda Terminal – Show Inventory

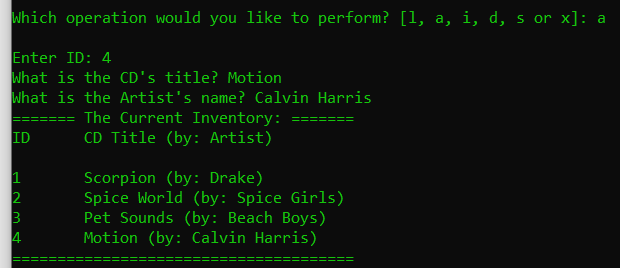


Figure 11 – Image of Working Script in Anaconda – Add CD

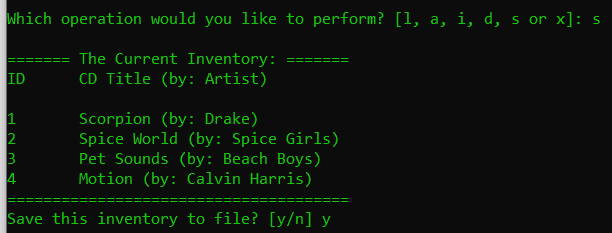


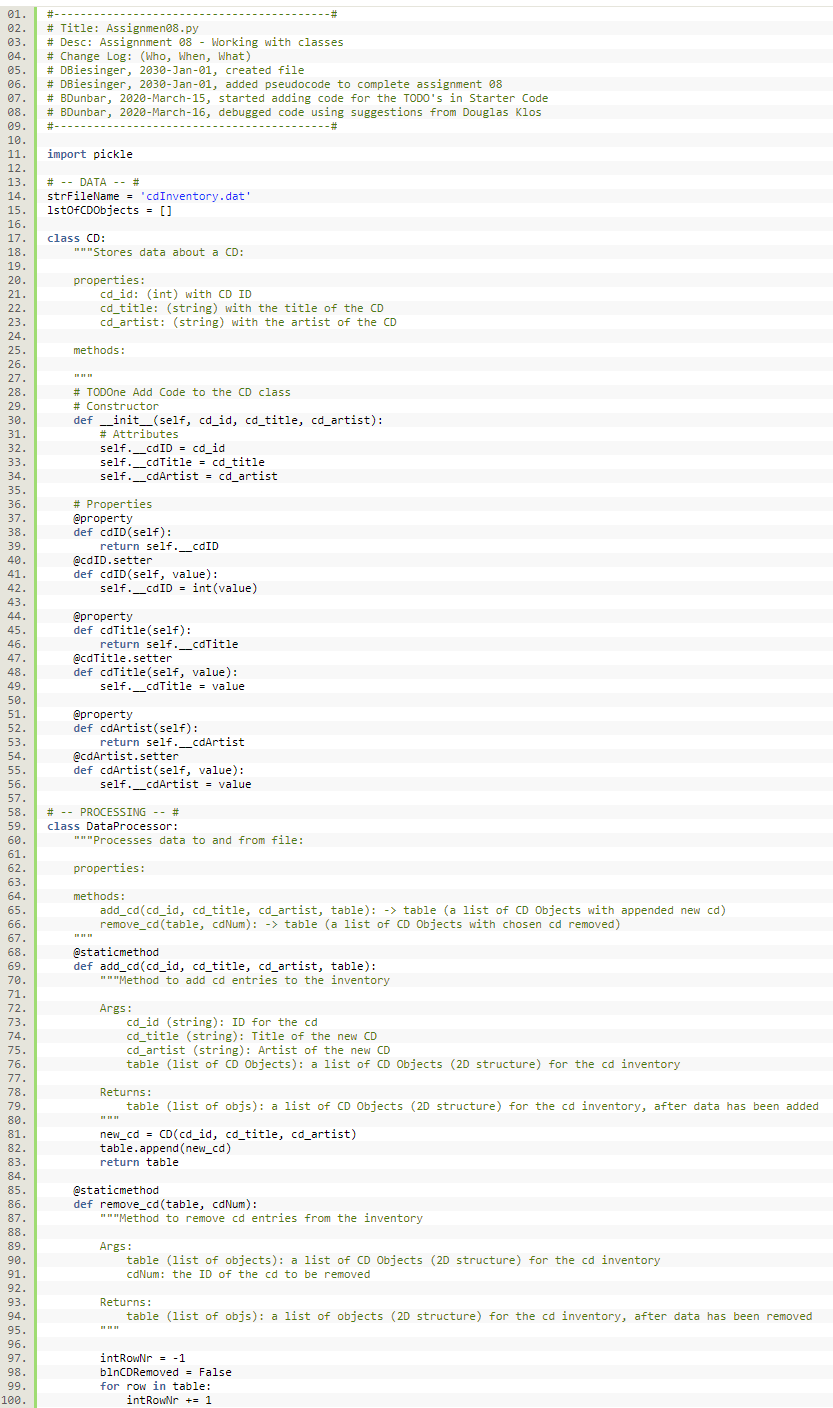
Figure 12 - Image of Working Script in Anaconda Terminal – Save Inventory

# GitHub

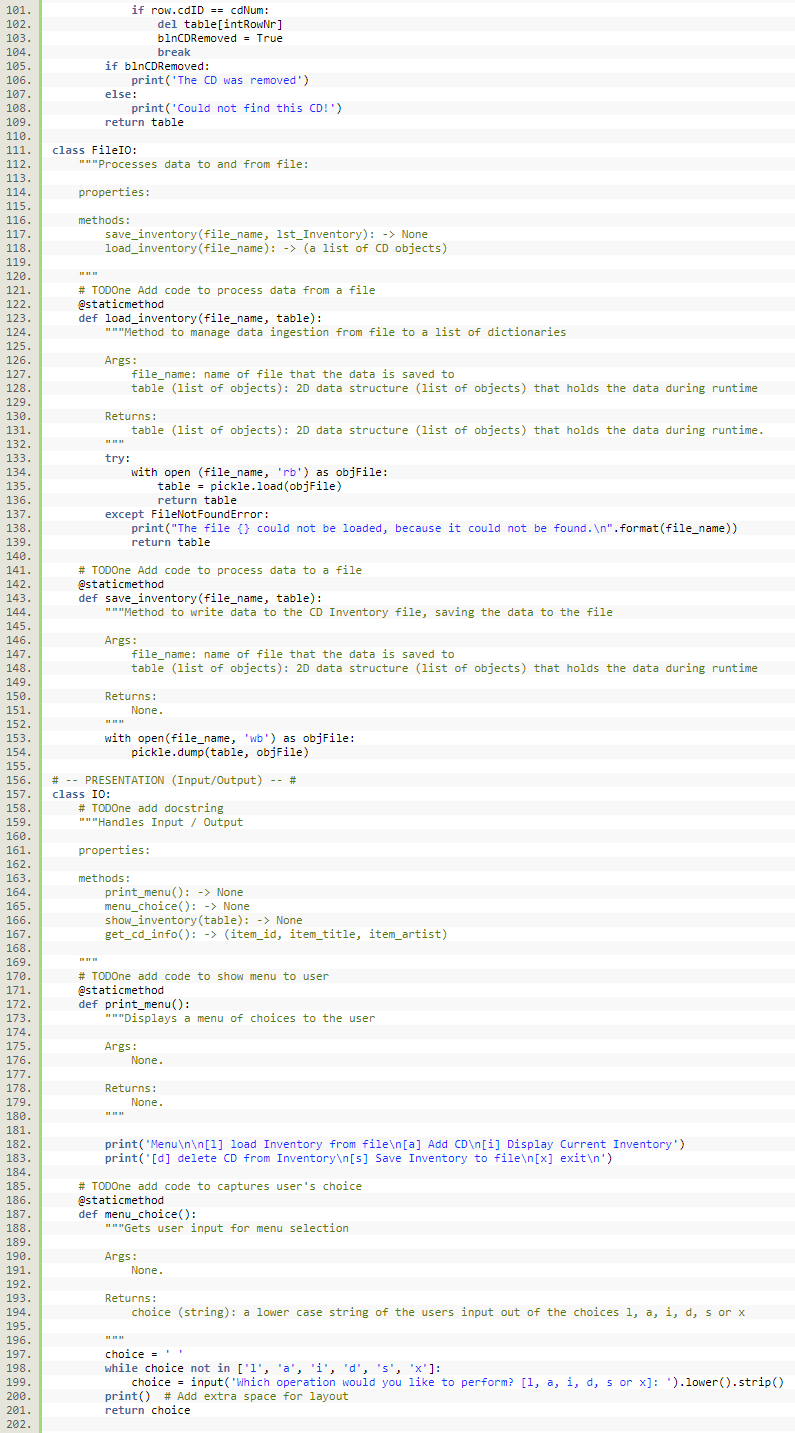
My assignment 08 files are uploaded to GitHub: <https://github.com/brynbar/Assignment_08>

# Appendix

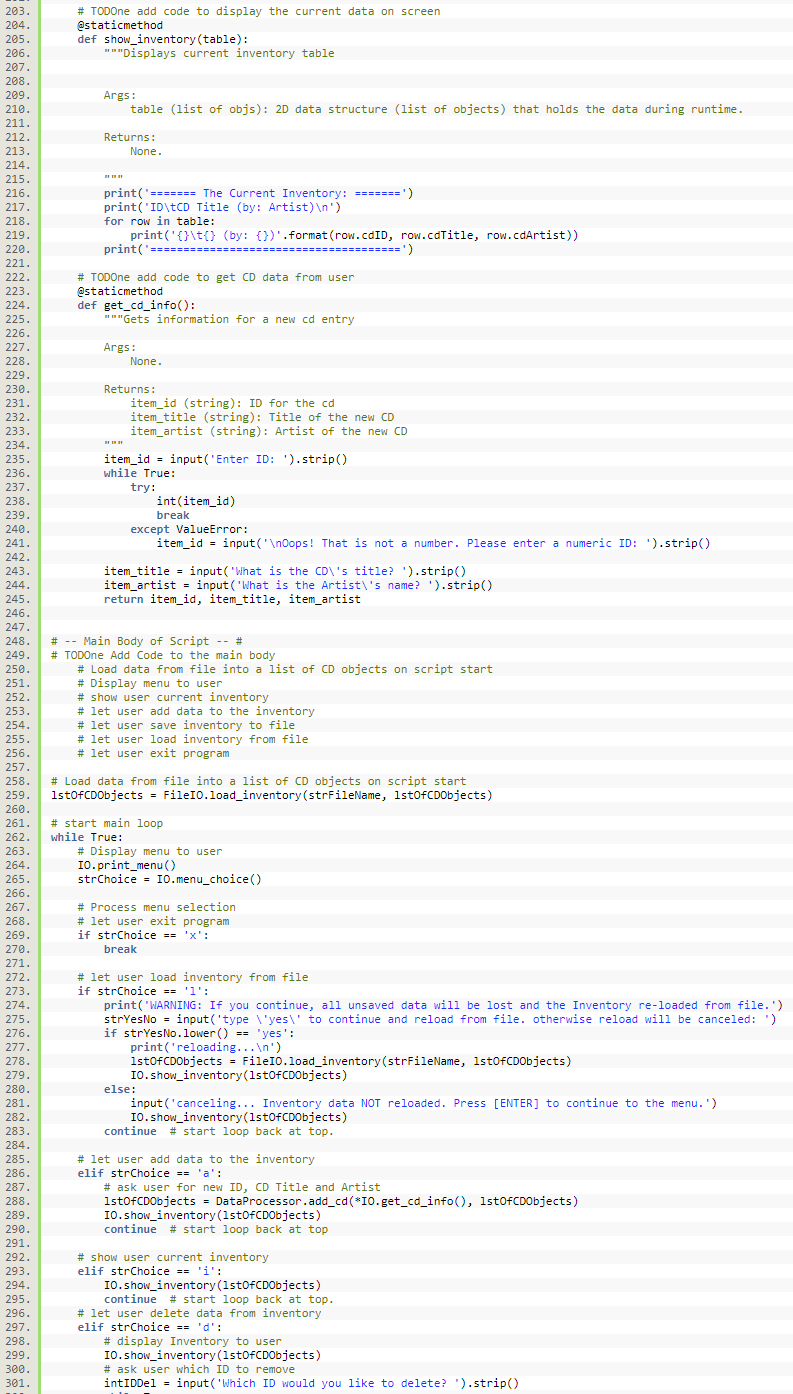
Using [PlanetB’s](http://www.planetb.ca/syntax-highlight-word) (external reference web page) [[1]](#footnote-1)



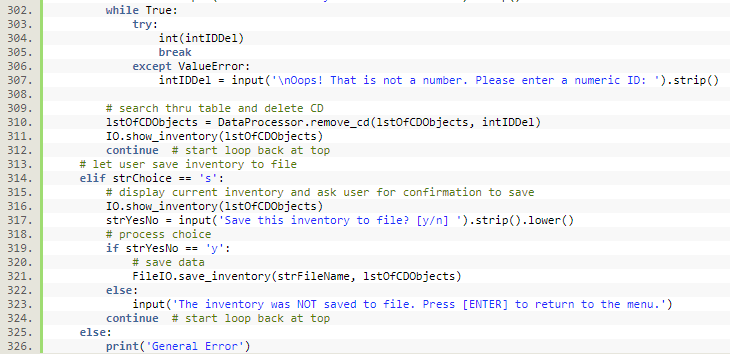
LISTING 1.0 – Source Code



LISTING 1.1 – Source Code



LISTING 1.2 – Source Code



Listing 1.3 - Source Code

1. <http://www.planetb.ca/projects/syntaxHighlighter/popup.php> - retrieved 16-March-20 [↑](#footnote-ref-1)