Danielle Rodriguez

24 March 2022

IT FDN 110 B

Assignment 08

Object Oriented Programming

Introduction

This module introduced software objects and Object Oriented Programming (OOF). This is a complex topic which takes a few go arounds to fully understand. In Object Oriented Programming, everything is an object which has attributes and methods. OOF utilizes classes and objects, fields, constructors, attributes and methods to build programs. This week’s lesson introduces the basics of OOF, which will be built upon in following modules.

The Building Blocks

Text

Description automatically generatedClasses package and serve as a blueprint for the object. The class holds the fields, constructor, attributes, properties, and methods. In Figure 1, a class is created called TrackInfo which stores CD information. This code only contains one field which is then referenced later in the code. Fields are created in a similar way as variables and store the data in a class. Figure 1 shows how fields can be used to store various types of data, including integers and strings.

**Figure 1: Using Fields**

Constructors is a specific method that is used when creating an object. Figure 2 shows how constructors can be used in a program. This code uses Python’s constructor method **\_\_init\_\_()** which allows the programmer to call Class like a function. Figure 2 also uses the keyword **self**, which is the first parameter used in every method.

Text

Description automatically generated

**Figure 2: Using Constructors**

Text

Description automatically generatedAttributes are also useful when working with Object Oriented Functions. Attributes are fields within the program that hold data which can sometimes be tricky as the programmer has little control over what goes into the attributes. Figure 3 outlines an example using attributes, building off the pervious figures. In the example below, the field is blank and not referenced in the program. The print functions call the message attribute to display the track information.

**Figure 3: Using with Attributes**

Properties are another useful tool within Object Oriented Functions. There are often times two properties created for each attribute referred to as the ‘getter’ and ‘setter’. The getter allows the programmer to format the fields while setter methods allow for error handling.

Figure 4 further builds on the previous examples, adding properties. The code is also modified, including adding separate fields and attributes for each variable. This allowed me to create ‘getter’ properties for each variable. In this example, only one setter is used to validate the ID number is an integer. The final lines of the code differ from the previous examples, defining the values in one line.

Text

Description automatically generated

**Figure 4: Using Properties**

The Modified Code

Using Object Oriented Functions, I was able to modify the CD Inventory program. This program uses the tool introduced in this module as seen in the first half of the code. Notably, the properties section defines each variable using a ‘getter’ and provides error handling through the ‘setter’. This code also utilizes methods, which help format the data. In this case, the code prints the data to the user and file in a nicely formatted manner.

Text

Description automatically generated

This module was straightforward as most of the code came from last week’s module. But I found myself getting caught up by minor syntax errors. Using parts of last week’s model was challenging as I had to change some functions (pickling) and make sure the variable names, file names and other data labels matched across the different sources.

Summary

This module introduced the basis to Object Oriented Functions, building a foundation for next week’s material. This lesson outlined how to use classes and the various attributes within classes. OOF is a complex topic which requires extensive work to fully understand how powerful the function can be. This module can be found in GitHub.

Appendix

Text

Description automatically generated

**Appendix 1: CD Inventory code in Spyder**

Text

Description automatically generated

**Appendix 2: CD Inventory code in Terminal**

*#------------------------------------------#*

*# Title: CD\_Inventory.py*

*# Desc: Assignnment 08 - Working with classes*

*# Change Log: (Who, When, What)*

*# DBiesinger, 2030-Jan-01, created file*

*# DBiesinger, 2030-Jan-01, added pseudocode to complete assignment 08*

*# DRodriguez, 2021-Mar-24, modified code*

*#------------------------------------------#*

*# -- DATA -- #*

file\_name = 'cdInventory.txt'

table = []

**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:*

*"""*

*# -- Fields -- #*

*# -- Constructor -- #*

**def** \_\_init\_\_(self, cd\_id, cd\_title, cd\_artist):

*# -- Attributes -- #*

**try**:

self.\_\_cd\_id = int(cd\_id)

self.\_\_cd\_title = str(cd\_title)

self.\_\_cd\_artist = str(cd\_artist)

**except** **ValueError**:

print('Please enter a valid CD ID.')

**except** **Exception** **as** e:

**raise** **Exception**('ERROR: Error defining CD inputs')

*# -- Properties -- #*

@property

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

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

@cd\_id.setter

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

**try**:

self.\_\_cd\_id == int(value)

**except** **ValueError**:

print('Please enter a valid CD ID.')

@property

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

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

@cd\_title.setter

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

**try**:

self.\_\_cd\_title == str(value)

**except** **Exception** **as** e:

**raise** **Exception**('ERROR: Please enter a CD title as a string.')

@property

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

**return** self.\_\_cd\_artist

@cd\_artist.setter

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

**try**:

self.\_\_cd\_artist == str(value)

**except** **Exception** **as** e:

**raise** **Exception**('ERROR: Please enter a Artist as a string.')

*# -- Methods -- #*

**def** print\_to\_screen(self):

*"""prints data to screen"""*

**return** '**{:>2}\t{}** (by **{}**)'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)

**def** print\_to\_file(self):

*"""prints data to file"""*

**return** '**{}**,**{}**,**{}\n**'.format(self.\_\_cd\_id, self.\_\_cd\_title, self.\_\_cd\_artist)

*# TODO Add Code to the CD class*

**pass**

*# -- PROCESSING -- #*

**class** **FileIO**:

*"""Processes data to and from file:*

*properties:*

*methods:*

*save\_inventory(file\_name, lst\_Inventory): -> None*

*load\_inventory(file\_name): -> (a list of CD objects)*

*"""*

*# TODO Add code to process data from a file*

*# TODO Add code to process data to a file*

**pass**

*# -- PRESENTATION (Input/Output) -- #*

**class** **IO**:

@staticmethod

**def** print\_menu():

*"""Displays a menu of choices to the user*

*Args:*

*None.*

*Returns:*

*None.*

*"""*

print('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')

print('[s] Save Inventory to file**\n**[x] exit**\n**')

@staticmethod

**def** menu\_choice():

*"""Gets user input for menu selection*

*Args:*

*None.*

*Returns:*

*choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x*

*"""*

choice = ' '

**while** choice **not** **in** ['l', 'a', 'i', 's', 'x']:

choice = input('Which operation would you like to perform? [l, a, i, s or x]: ').lower().strip()

print() *# Add extra space for layout*

**return** choice

@staticmethod

**def** show\_inventory(table):

*"""Displays current inventory table*

*Args:*

*table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.*

*Returns:*

*None.*

*"""*

print('======= The Current Inventory: =======')

print('ID**\t**CD Title (by: Artist)**\n**')

**for** objCD **in** table:

print(objCD.print\_to\_screen())

print('======================================')

@staticmethod

**def** add\_CD():

*"""Ask user for new ID, CD Title and Artist*

*Args:*

*strID: name of ID input*

*strTitle: name of title input*

*strArtist: name of artist input*

*intID: Converts strID to integer*

*Returns:*

*values defined in arguments if no errors occur*

*"""*

**while** **True**:

strID = input('Enter ID: ').strip()

**try**:

intID = int(strID)

**break**

**except** **ValueError**:

print('Please enter a number for CD ID.')

**except** **Exception** **as** e:

print('Request could not be completed. There was an error.')

print(type(e), e, e.\_\_doc\_\_, sep='**\n**')

strTitle = input('What is the CD**\'**s title? ').strip()

strArtist = input('What is the Artist**\'**s name? ').strip()

intID = int(strID)

**return** intID, strTitle, strArtist

*# TODO add docstring*

*# TODOne add code to show menu to user*

*# TODOne add code to captures user's choice*

*# TODOne add code to display the current data on screen*

*# TODOne add code to get CD data from user*

**pass**

**class** **FileProcessor**:

*"""Processing the data to and from text file"""*

@staticmethod

**def** read\_file(file\_name, table):

*"""Function to manage data ingestion from file to a list of dictionaries*

*Reads the data from file identified by file\_name into a 2D table*

*(list of dicts) table one line in the file represents one dictionary row in table.*

*Args:*

*file\_name (string): name of file used to read the data from*

*table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*

*Returns:*

*Table if no errors occur*

*"""*

table.clear() *# this clears existing data and allows to load data from file*

**try**:

**with** open(file\_name, 'r') **as** file:

**for** line **in** file:

cd\_id, cd\_title, cd\_artist = line.strip(',').split()

objCD = CD(cd\_id, cd\_title, cd\_artist)

table.append(objCD)

**except** **FileNotFoundError**:

print('File not found.')

**except** **Exception** **as** e:

print('Request could not be completed. There was an error.')

print(type(e), e, e.\_\_doc\_\_, sep='**\n**')

**else**:

**return** table

**class** **DataProcessor**:

@staticmethod

**def** add\_CD(new\_CD, lstTbl):

*"""Function to add a new item to table*

*Args:*

*dicRow: creates dictionary*

*lstTbl: adds new values to dictionary*

*show\_inventory: displays data*

*Returns:*

*Table showing inventory in lstTbl*

*"""*

intID, strTitle, strArtist = new\_CD

new\_CD = CD(intID, strTitle, strArtist)

table.append(new\_CD)

**return** table

@staticmethod

**def** write\_file(file\_name, table):

*"""Function to save data*

*Args:*

*objFile: calls the text file containing CD inventory*

*lstValues: row of values in 2D table*

*objFile.write: writes inputted data to file*

*stTbl (list of dict): 2D data structure (list of dicts) that holds the data during runtime.*

*Returns:*

*Table showing inventory in lstTbl if no errors occur*

*"""*

**try**:

**with** open(file\_name, 'w') **as** readfileObj:

**for** objCD **in** table:

file\_name.write(objCD.print\_to\_file())

**except** **FileNotFoundError**:

print('File not found.')

**except** **Exception** **as** e:

print('Request could not be completed. There was an error.')

print(type(e), e, e.\_\_doc\_\_, sep='**\n**')

*# -- Main Body of Script -- #*

*# TODO Add Code to the main body*

*# Load data from file into a list of CD objects on script start*

*# Display menu to user*

*# show user current inventory*

*# let user add data to the inventory*

*# let user save inventory to file*

*# let user load inventory from file*

*# let user exit program*

**while** **True**:

*# Display Menu to user and get choice*

IO.print\_menu()

strChoice = IO.menu\_choice()

*# Process menu selection*

*# Process exit first*

**if** strChoice == 'x':

**break**

*# Load inventory*

**if** strChoice == 'l':

print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')

strYesNo = input('type **\'**yes**\'** to continue and reload from file. otherwise reload will be canceled')

**if** strYesNo.lower() == 'yes':

print('reloading...')

FileProcessor.read\_file(file\_name, table)

IO.show\_inventory(table)

**else**:

input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')

IO.show\_inventory(table)

**continue** *# start loop back at top.*

*# Add a CD*

**elif** strChoice == 'a':

*# Ask user for new ID, CD Title and Artist and add item to table*

new\_cd = IO.add\_CD()

lstTbl = DataProcessor.add\_CD(new\_cd, table)

**continue** *# start loop back at top.*

*# Display current inventory*

**elif** strChoice == 'i':

IO.show\_inventory(lstTbl)

**continue** *# start loop back at top.*

*# Save inventory to file*

**elif** strChoice == 's':

*# Display current inventory and ask user for confirmation to save*

IO.show\_inventory(lstTbl)

strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()

*# Process choice*

**if** strYesNo == 'y':

*# Save data*

DataProcessor.write\_file(file\_name, table)

**else**:

input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

**continue** *# start loop back at top.*

*#Catch-all should not be possible, as user choice gets vetted in IO, but to be save:*

**else**:

print('General Error')

**Appendix 3: CD Inventory code**