**Mitchell Post**

**FEB-18-2020**

**IT FDN 110 A**

**MOD 6**

# **Introduction**

This week we covered many concepts mostly focusing on variables, classes, and functions and how they all relate. We spent a lot of time discussing parameters and arguments. How I understand them so far is parameters and arguments are both information passed into a function. While parameter can be more of an empty variable waiting for a value, and argument has a value to be passed into the function. Whether as a result of a function in a “Return value” or defined somewhere else, through user input or variable definitions.

Another concept to understand this week is variable association whether it be local or global. A global variable is assigned either within a function, as a defined global variable, or somewhere else in the main body of the code. This variable as it name implies is global and whenever called out it will equal its assigned value. A local variable is only its given value within a function, as a return value. On top of that there are “shadow values” that are global values that have been given separate local values within a function.

Third concept we went over this week was understanding the utility of functions within code, class structure, and separation of concerns (SOC). The utility of functions is that they give you the ability to call out jobs instead of re writing code multiple times. Also, functions can pass in different data making them more dynamic when used for handling different data objects vice writing out the entire code each time and having to insert the data arguments/parameters manually. Class structure is to organize grouping like type purpose, functions together So far from my brief exposure to Python it looks like this is only for organization at this point. I’m sure as you get into more advance coding there is utility to class structure as well. Separation of concerns is the structure of which your functions are organized. As I mentioned you organize your function by purpose, for instance in this code user input/Data handling/Processing all these types have their own class, that organization is SoC.

# **Assignment**

* Using provided CDInventory.py starter perform the following tasks
  + Modify the starter utilizing functions with SoC in mind
  + Complete several TODOs involving inserting/re-arranging functions
  + Test Functionality of program as whole

**Modify the starter utilizing functions with SoC in mind**

The first thing I did was look at what code I could take and turn into functions. I made a list of all of them and tagged them a name related to what they were doing to understand the program. I then looked at what classes were provided and decided where I would put each function. After I got all this organized, I started taking the code from only one section at a time and creating a function for it. I then tested the function within the program to ensure it worked properly, before moving to the next. I submitted my code for review after this and I found that my SoC organization wasn’t quite correct. All the comments made sense and I believe I have now re-arranged the code correctly.

**TODOs inserting/re-arranging functions**

These tasks seemed to be fairly straight forward; it was vital to ensure the right data was being passed into each function while creating them. I ran into a few issues not having the correct value inserted as I worked on my code. The below function Figure 1 I found to be the most interesting this I/O function takes the input of from the user then returns the values to the nested function witch then appends the table for the program. \*Update\* after class I removed my nested append function. I now defined input user as the three pieces of CD info and passed that into my append table. Figure 2.

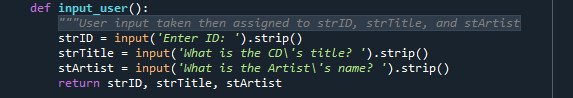


Figure 1 – CD input function

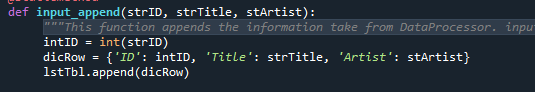


Figure 2 – Append lstTble

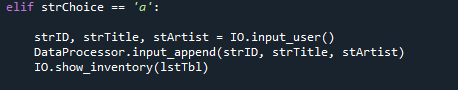
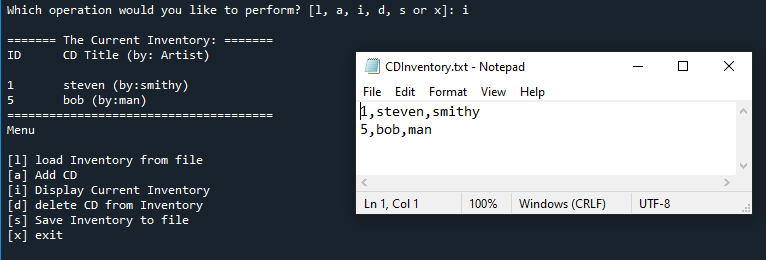


Figure 3 – Assigning strID. strTitle, stArtist and passing to append function

**Test Functionality of program as whole**

After all was done and re-organized the program seems to run as intended. I think it was very helpful using the approach I took. First organizing and understanding the program and code, then organizing a plan to re-arrange then code. Also testing each function one at a time before creating the next. If I didn’t take this approach. I could see this assignment becoming very difficult.



*Figure 3 – Testing Functionality*

**References**

Dawson, M. (2009). *Python® Programming for the Absolute Beginner, Third Edition*. Course Technology PTR.

# **Appendix**

#------------------------------#

#------------------------------------------#

# Title: Assignment06\_Starter.py

# Desc: Working with classes and functions.

# Change Log: (Who, When, What)

# POST, 0800-FEB-15, Created File

# POST, 0900-FEB-15, pulled out code for functions and organized for task

# POST, 1100-FEB-15, assigned all code to defintions according to appropriate class

# POST, 1200-FEB-15, troubleshot nested code append

# POST, 1300-FEB-15, Tested code succesfully

# POST, 0815-FEB17, RE ORG FUNCTIONS/CLASS

#------------------------------------------#

#------------------------------#

# -- DATA -- #

strChoice = '' # User input

lstTbl = [] # list of lists to hold data

dicRow = {} # list of data row

strFileName = 'CDInventory.txt' # data storage file

objFile = None # file object

value1 = None # is this neccessary? for "def delete\_cd(value1):" Do i have to none out variables

# -- PROCESSING -- #

class DataProcessor:

"""Processing the data from input to Dict"""

@staticmethod

def input\_append(strID, strTitle, stArtist):

"""This function appends the information take from DataProcessor. input\_user(strFileName, lstTbl)

then uses it to append dicrRow

Args:

dicRow: dictionary row.

Table (List of dicts)

Returns:

None

"""

intID = int(strID)

dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}

lstTbl.append(dicRow)

IO.show\_inventory(lstTbl)

@staticmethod

def delete\_cd(value1):

""" Deletion function added here to I/o class

Args:

None.

Returns:

intIDDel User input to select ID to be deleted

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

User input is required to delete desired CD"""

intRowNr = -1

blnCDRemoved = False

for row in lstTbl:

intRowNr += 1

if row['ID'] == intIDDel:

del lstTbl[intRowNr]

blnCDRemoved = True

break

if blnCDRemoved:

print('The CD was removed')

else:

print('Could not find this CD!')

# -- File Proccessing -- #

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:

None.

"""

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

objFile = open(file\_name, 'r')

for line in objFile:

data = line.strip().split(',')

dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}

table.append(dicRow)

objFile.close()

@staticmethod

def write\_file(strFileNameame, lstTbl):

""" Additional file proccesing function to save file to .txt

Function uses values of strFilName(.txt storage target) and dict list lstbl"""

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

if strYesNo == 'y':

objFile = open(strFileName, 'w')

for row in lstTbl:

lstValues = list(row.values())

lstValues[0] = str(lstValues[0])

objFile.write(','.join(lstValues) + '\n')

objFile.close()

else:

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

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

class IO:

"""Handling Input / Output"""

@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('[d] delete CD from Inventory\n[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', 'd', 's', 'x']:

choice = input('Which operation would you like to perform? [l, a, i, d, 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\tCD Title (by: Artist)\n')

for row in table:

print('{}\t{} (by:{})'.format(\*row.values()))

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

@staticmethod

def input\_user(strFileName, lstTbl):

"""User input taken then assigned to strID, strTitle, and stArtist

Nested function Io.input\_append used to take local variable then

Appended to DICT list

Args:

IO.input\_append(strID, strTitle, stArtist): Function used to take read and append local returns

Returns:

strID = input ('Enter ID: ')

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

stArtist = input ('What is the Artist\'s name? ')

"""

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

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

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

DataProcessor.input\_append(strID, strTitle, stArtist)

#### When program starts, read in the currently saved Inventory (no change)

FileProcessor.read\_file(strFileName, lstTbl)

##### start main loop ### (no change)

while True:

IO.print\_menu()

strChoice = IO.menu\_choice()

if strChoice == 'x':

break

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(strFileName, lstTbl)

IO.show\_inventory(lstTbl)

else:

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

IO.show\_inventory(lstTbl)

continue # start loop back at top (No Change)

elif strChoice == 'a':

IO.input\_user(strFileName, lstTbl)

continue # start loop back at top.

elif strChoice == 'i':

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

continue # start loop back at top.

elif strChoice == 'd':

intIDDel = int(input('Which ID would you like to delete? ').strip())

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

DataProcessor.delete\_cd(lstTbl) ## Calls function to delete desired CD

continue # start loop back at top.

elif strChoice == 's':

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

FileProcessor.write\_file(strFileName, lstTbl) ## Calls for to save inventory to target .txt file

continue # start loop back at top.

### catch-all should not be possible, as user choice gets vetted in IO, but to be save (no change):

else:

print('General Error')