Troy McFarland

2021-02-21

Foundations of Programming, Python

Assignment\_06

Git Hub: https://github.com/DJ-TMC/Assignment\_06

# Modfying Another Script by Adding Functions

# Introduction

In this paper, I’ll be going over the sixth set of lessons for the Foundations of Programming, Python class taught by Dirk Biesinge and assisted by Douglas Klos. I’ll go over things learned, and insights I had coding longer programs in Spyder while working on assignment 6.

# Overview of New Things Learned in Module 6

* What is a function?
  + A way to group lines of code together so they can be used elsewhere in the program, and more than once
  + Similar to a ‘macro’ in other programs
* What are parameters?
  + The inputs declared in a function
* What are arguments?
  + The actual values that is being passed through to a parameter via the parameters
* What is the difference between parameters and arguments?
  + Minor difference, but the arguments are the actual values, while the parameters are the inputs declared in a function. <https://developer.mozilla.org/en-US/docs/Glossary/Parameter> [[1]](#footnote-1) (external link)
* What are return values?
  + What a function returns to the main program, either as a variable, or instantly, such as in a print statement.
* What is the difference between a global and a local variable?
  + A global variable can be accessed by the entire program. A local variable can only be accessed by the function in which it resides.
* What is shadowing?
  + The act of creating a variable name that is identical inside a function to a variable outside the function or elsewhere in the program. Because the shadow variable can hold different values, it can create confusion when debugging or working with a team. Therefore, they generally should be avoided.
* How do you use functions to organize your code?
  + By calling functions from a main program, you can put similar functions together (see below for more detail)
  + Also, an advantage of functions is that they are encapsulated, and once a piece is working properly, one can generally leave them alone while trouble shooting other parts of the code. This is especially handy when a team is working on the same program.
* What is the difference between a function and a class?
  + Classes are ways of grouping functions (and variables and constants) together.
* How do functions help you program using the “Separations of Concerns" pattern?
  + By calling functions from a main program, a programmer can fairly easily separate their code into Data, Processing, and Presentation sections, or whatever organizational method works best for a team.

# Coding in Python

I’m now getting to the point to where I can see more than one way top solve a problem. For example, part of assignment 6 was to move the input of one section of code to the IO class, and the processing section to the DataProcessor class. The challenge for me was figuring out how to best get the data from the input and move it into the Data Processor. There are three ways I can think of on how to do this:

1. Call the **DataProcessor.append\_table** function from the **IO.user\_cd\_input** function.
   1. Might work with a small program like this, but this circumvents the main part of the program, so another programmer looking at this code might not understand how it flows.
2. Create a global variable list that the input updates, as shown in Listing 14 of FDN\_Py\_Module06, by assigning it as an argument to the function. Because a list holds references, they would update when any change is made to them, and the data processer takes in as arguments
   1. I almost went this route. But, I saw in the book that while this is possible, it is not always recommended. I think this would be a little easier for another programmer to understand than the first method but it doesn’t seem very safe.
3. I went with this third way: Get the return from the **IO.user\_cd\_input** function back in the main loop, and then assign them as arguments when calling the **DataProcessor.append\_table** function. I struggled with the syntax a bit longer than expected. I slept on it and when I got up the next morning, I had an idea. Happy to see that it worked! I remembered that the flow for an equal sign in Python is reverse to what I’m used to. So, I assigned the variables to the return of **IO.user\_cd\_input**
   1. Positives: This is clearer for other programmers to read.
   2. Negatives: It requires a few more variable names to handle the hand off and prevent shadow variables.
   3. Below is the code as it currently stands (I may update variable names before submission)
4. *# 3.3 ADD CD process add a CD*
5. elif strChoice **==** 'a'**:**
6. *#assign received data from user variables from return of IO.user\_cd\_input*
7. recStrId**,** recStrTitle**,** recStrArtist **=** IO**.**user\_cd\_input**()**
8. *#Feed received data from user into DataProcessor.append\_table*
9. DataProcessor**.**append\_table**(**recStrId**,** recStrTitle**,** recStrArtist**)**
10. continue *# start loop back at top.*

Listing : The hardest nut for me to crack for this assignment.

# Additional Things Learned

Here’s a few additional; things I learned while working on this week’s assignments and labs:

* Spyder has split view abilities!
  + This enables one to see the top and bottom of their code easier
  + It also enables the programmer to see other scripts in other tabs at the same time
    - Caution: Combined makes for the possibility that the user is editing the wrong program. (I did this more than once)
* Sypder has the ability to collapse code based on functions. So, if the knows that p[art of the code is solid, they can collapse it to view the other parts of their code easier.
* Spyder’s highlight function has been helpful in the past, but it is much more helpful with larger scripts.
  + I double checked to make sure all my functions were being used elsewhere in the script
    - There was one extra and unnecessary function that I brought in from lesson 5. By checking function names, I found it. It was no longer plugged into any other part of the program, as Dirk’s code was solving it’s use, so I deleted it.

# Assignment 06 Running in Spyder:

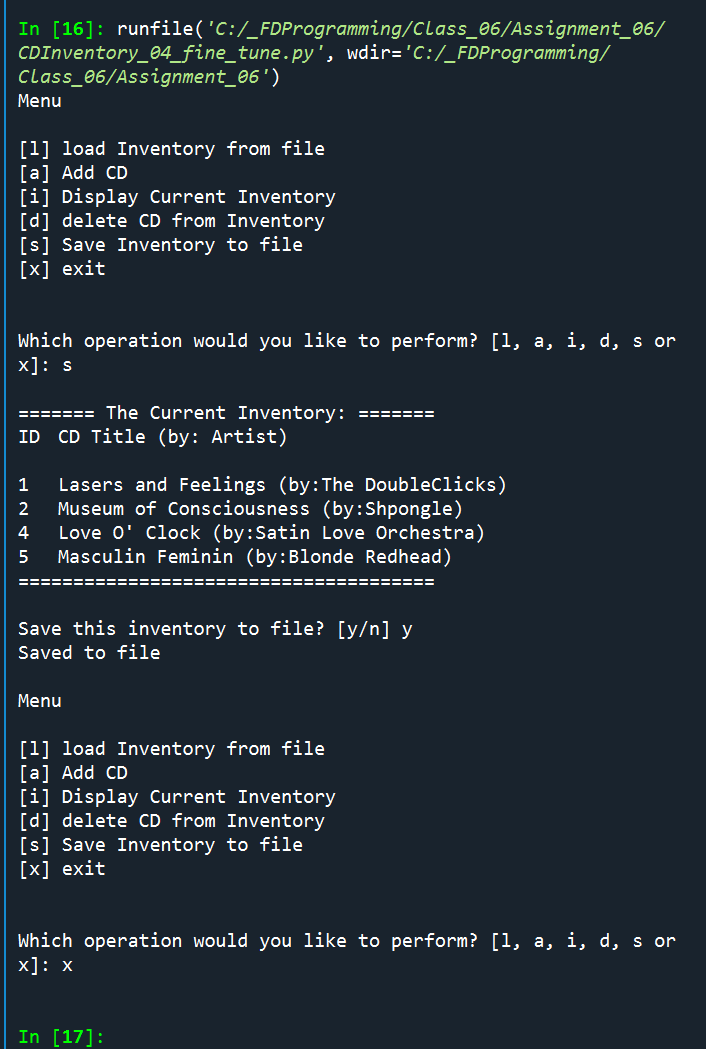


Figure : Example of Program running in Spyder

# Assignment 06 Running in the Anaconda Terminal:

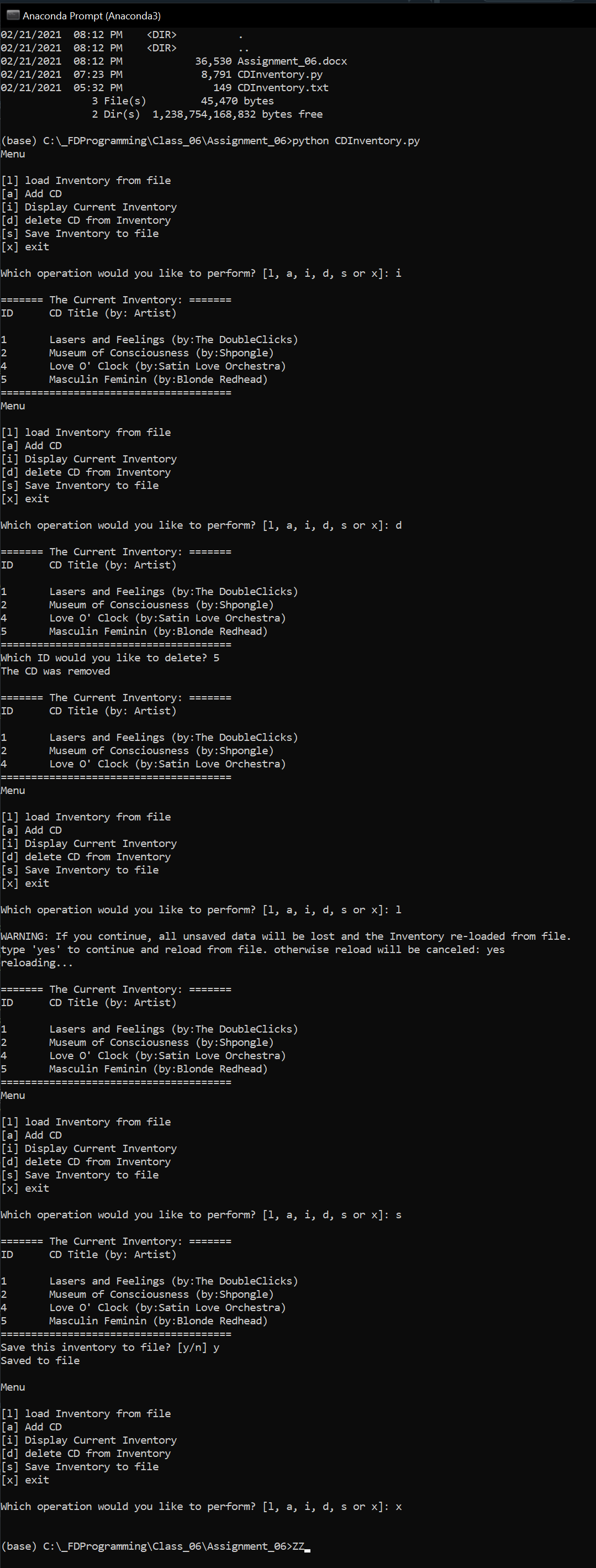


Figure : example of program running in terminal

# Summary

In this paper, I described things learned in the sixth set of lessons for the Foundations of Programming, Python class taught by Dirk Biesinge and assisted by Douglas Klos. In addition to the expected things to learn for this assignment, I went over my eureka moment for figuring out how to move values around between various functions without creating shadow variables, and keeping the flow as safe as possible for encapsulation and for other team members to work on.

# Appendix

Full Code of Assignment 6:

1. *#------------------------------------------#*
2. *# Title: Assignment06\_Starter.py*
3. *# Desc: Working with classes and functions.*
4. *# Change Log: (Who, When, What)*
5. *# DBiesinger, 2030-Jan-01, Created File*
6. *# TMcFarland, 2021-Feb-19, started mods: adding functions, looking at TODOs*
7. *#------------------------------------------#*
8. *#Importing os.path for additional FileProcessor.read\_file functionality, to check to see if text file exists*
9. import os**.**path
10. *# -- DATA -- #*
11. strChoice **=** '' *# User input*
12. lstTbl **=** **[]** *# list of dicts to hold data*
13. dicRow **=** **{}** *# dict of data row*
14. strFileName **=** 'CDInventory.txt' *# data storage file*
15. objFile **=** None *# file object*
16. *# -- PROCESSING -- #*
17. class DataProcessor**:**
18. @staticmethod
19. def delete\_dict**(**delDict**):**
20. """Deletes lstTbl dicitonary entry based on user input of ID number
21. Args:
22. delDict (integer): ID number representing table entry
23. Returns:
24. NONE
25. """
26. intRowNr **=** **-**1
27. blnCDRemoved **=** False
28. for row in lstTbl**:**
29. intRowNr **+=** 1
30. if row**[**'ID'**]** **==** delDict**:**
31. del lstTbl**[**intRowNr**]**
32. blnCDRemoved **=** True
33. break
34. if blnCDRemoved**:**
35. print**(**'The CD was removed\n'**)**
36. else**:**
37. print**(**'Could not find this CD!\n'**)**
38. @staticmethod
39. def append\_table**(**strID**,** strTitle**,** strArtist**):**
40. *# 3.3.2 Add item to the table*
41. """Takes arguments from Main loop append section and appends to lstTbl list of dictionaries in memory
42. Args:
43. strID (string): ID number
44. strTitle (string): CD title
45. StrArtist (string); Artist Name
46. Returns:
47. None
48. """
49. intID **=** int**(**strID**)**
50. dicRow **=** **{**'ID'**:** intID**,** 'Title'**:** strTitle**,** 'Artist'**:** strArtist**}**
51. lstTbl**.**append**(**dicRow**)**
52. IO**.**show\_inventory**(**lstTbl**)**
53. class FileProcessor**:**
54. """Processing the data to and from text file"""
55. @staticmethod
56. def read\_file**(**file\_name**,** table**):**
57. """Function to manage data ingestion from file to a list of dictionaries
58. Reads the data from file identified by file\_name into a 2D table
59. (list of dicts). One line in the file represents one dictionary row in table.
60. Args:
61. file\_name (string): name of file used to read the data from
62. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
63. Returns:
64. None.
65. """
66. table**.**clear**()** *# this clears existing data and allows to load data from file*
67. *#check to see if database file already exists. If not, create blank one.*
68. if os**.**path**.**exists**(**'CDInventory.txt'**):**
69. *#print('yay! already exists')*
70. objFile **=** open**(**file\_name**,** 'r'**)**
71. for line in objFile**:**
72. data **=** line**.**strip**().**split**(**','**)**
73. dicRow **=** **{**'ID'**:** int**(**data**[**0**]),** 'Title'**:** data**[**1**],** 'Artist'**:** data**[**2**]}**
74. table**.**append**(**dicRow**)**
75. objFile**.**close**()**
76. else**:**
77. *#create blank database in same folder as script*
78. objFile **=** open**(**strFileName**,** 'w'**)**
79. objFile**.**close**()**
80. def write\_file**():**
81. """Writes List of Dicitonaries lstTbl from memory into a text file.
82. Ensures proper comma seperated formatting for best storage and retrieval
83. Args:
84. None
85. Returns:
86. None
87. """
88. *# TODO move processing code into function*
89. objFile **=** open**(**strFileName**,** 'w'**)**
90. for row in lstTbl**:**
91. lstValues **=** list**(**row**.**values**())**
92. lstValues**[**0**]** **=** str**(**lstValues**[**0**])**
93. objFile**.**write**(**','**.**join**(**lstValues**)** **+** '\n'**)**
94. objFile**.**close**()**
95. print**(**'Saved to file\n'**)**
96. *# -- PRESENTATION (Input/Output) -- #*
97. class IO**:**
98. """Handling Input / Output"""
99. @staticmethod
100. def print\_menu**():**
101. """Displays a menu of choices to the user
102. Args:
103. None.
104. Returns:
105. None.
106. """
107. print**(**'Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory'**)**
108. print**(**'[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n'**)**
109. @staticmethod
110. def menu\_choice**():**
111. """Gets user input for menu selection
112. Args:
113. None.
114. Returns:
115. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
116. """
117. choice **=** ' '
118. while choice not in **[**'l'**,** 'a'**,** 'i'**,** 'd'**,** 's'**,** 'x'**]:**
119. choice **=** input**(**'Which operation would you like to perform? [l, a, i, d, s or x]: '**).**lower**().**strip**()**
120. print**()** *# Add extra space for layout*
121. return choice
122. @staticmethod
123. def show\_inventory**(**table**):**
124. """Displays current inventory table
125. Args:
126. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
127. Returns:
128. None.
129. """
130. print**(**'======= The Current Inventory: ======='**)**
131. print**(**'ID\tCD Title (by: Artist)\n'**)**
132. for row in table**:**
133. print**(**'{}\t{} (by:{})'**.**format**(\***row**.**values**()))**
134. print**(**'======================================'**)**
135. @staticmethod
136. def user\_cd\_input**():**
137. """ Receives user input for CD ID number, Album Title, and Artist Name
138. Args:
139. None
140. Returns:
141. All strings: CD ID number, CD Title, CD Artist Name
142. """
143. *# 3.3.1 Ask user for new ID, CD Title and Artist*
144. strID **=** input**(**'Enter ID: '**).**strip**()**
145. strTitle **=** input**(**'What is the CD\'s title? '**).**strip**()**
146. strArtist **=** input**(**'What is the Artist\'s name? '**).**strip**()**
147. return strID**,** strTitle**,** strArtist
148. *# -- MAIN PROGRAM -- #*
149. *# 1. When program starts, read in the currently saved Inventory*
150. FileProcessor**.**read\_file**(**strFileName**,** lstTbl**)**
151. *# 2. start main loop*
152. while True**:**
153. *# 2.1 Display Menu to user and get choice*
154. IO**.**print\_menu**()**
155. strChoice **=** IO**.**menu\_choice**()**
156. *# 3. Process menu selection*
157. *# 3.1 EXIT process exit first*
158. if strChoice **==** 'x'**:**
159. break
160. *# 3.2 LOAD process load inventory*
161. if strChoice **==** 'l'**:**
162. print**(**'WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.'**)**
163. strYesNo **=** input**(**'type \'yes\' to continue and reload from file. otherwise reload will be canceled: '**)**
164. if strYesNo**.**lower**()** **==** 'yes'**:**
165. print**(**'reloading...\n'**)**
166. FileProcessor**.**read\_file**(**strFileName**,** lstTbl**)**
167. IO**.**show\_inventory**(**lstTbl**)**
168. else**:**
169. input**(**'canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.\n'**)**
170. IO**.**show\_inventory**(**lstTbl**)**
171. continue *# start loop back at top.*
172. *# 3.3 ADD CD process add a CD*
173. elif strChoice **==** 'a'**:**
174. *#assign received data from user variables from return of IO.user\_cd\_input*
175. recStrId**,** recStrTitle**,** recStrArtist **=** IO**.**user\_cd\_input**()**
176. *#Feed received data from user into DataProcessor.append\_table*
177. DataProcessor**.**append\_table**(**recStrId**,** recStrTitle**,** recStrArtist**)**
178. continue *# start loop back at top.*
179. *# 3.4 DISPLAY INVENTORY process display current inventory*
180. elif strChoice **==** 'i'**:**
181. IO**.**show\_inventory**(**lstTbl**)**
182. continue *# start loop back at top.*
183. *# 3.5 DELETE process delete a CD*
184. elif strChoice **==** 'd'**:**
185. *# 3.5.1 get Userinput for which CD to delete*
186. *# 3.5.1.1 display Inventory to user*
187. IO**.**show\_inventory**(**lstTbl**)**
188. *# 3.5.1.2 ask user which ID to remove*
189. intIDDel **=** int**(**input**(**'Which ID would you like to delete? '**).**strip**())**
190. *# 3.5.2 search thru table and delete CD*
191. DataProcessor**.**delete\_dict**(**intIDDel**)**
192. IO**.**show\_inventory**(**lstTbl**)**
193. continue *# start loop back at top.*
194. *# 3.6 SAVE process save inventory to file*
195. elif strChoice **==** 's'**:**
196. *# 3.6.1 Display current inventory and ask user for confirmation to save*
197. IO**.**show\_inventory**(**lstTbl**)**
198. strYesNo **=** input**(**'Save this inventory to file? [y/n] '**).**strip**().**lower**()**
199. *# 3.6.2 Process choice*
200. if strYesNo **==** 'y' or 'yes'**:**
201. FileProcessor**.**write\_file**()**
202. *#consider an elif for 'n' using code below and re-ask question for user if they enter anything else*
203. else**:**
204. input**(**'The inventory was NOT saved to file. Press [ENTER] to return to the menu.'**)**
205. continue *# start loop back at top.*
206. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
207. else**:**
208. print**(**'General Error'**)**

1. Accessed 2021-02-21 [↑](#footnote-ref-1)