Brent Kieszling

8/19/2020

IT FDN 110: Introduction to Programming (Python)

Module 6 Learning Document

# Introduction

In this paper I will cover the standout points of Module 6. Some of the topics to expect are:defining functions, function returns, arguments, overloading functions, scopes, and doc string. Additionally, I will discuss notes on the development of the program I am submitting for Assignment 6.

# Lessons

A function is like a mini program. When called, it executes the block of code within it. Figure 1 below shows the function ‘getSum’ in action.

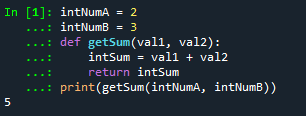


Figure - Example Function

This function is made up of three key parts: arguments, the code block, and the return statement. Arguments are assigned when defining the function, in ‘getSum’ they are ‘val1’ and ‘val2’. They are used to establish the number of expected inputs to the function and their order. The code block contains the work the function performs each time it is called. In ‘getSum’ the code block adds the two arguments together and assigns the value to ‘intSum’. The return statement is the information sent back to the main body of the code, in this case ‘intSum’ is returned. The final line of code in Figure 1 shows the function ‘getSum’ being called. Here, ‘intNumA’ and ‘intNumB’ are passed into the function as ‘val1’ and ‘val2’ respectively. Then, they are summed together and assigned to ‘intSum’. Finally, ‘intSum’ is returned to the ‘print()’ statement in the main body and the resulting output is 5. Functions can return multiple values, Appendix A.2 shows this. In Python, multiple returns from a function are packaged into a tuple.

A function lives in its’ own scope which is different than that of the main body. Scopes work like viewing points. When you move from the scope of the main body to that of a function it is like moving to a higher vantage point, Obi-Wan Kenobi might call this “taking the high ground”. You can see the things below you and some new stuff you couldn’t see at the lower point. Figure 2 shows this in more detail.

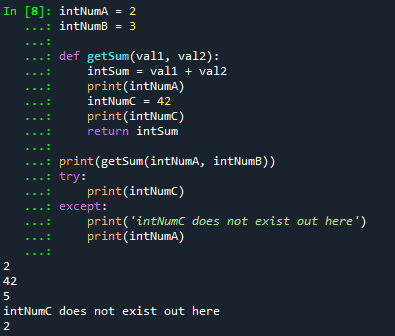


Figure - Scoping Things Out

First, notice that ‘intNumC’ is only defined in the function ‘getSum’. Since the function is at a higher scope ‘intNumC’ has not been defined in the main body by the time the ‘try’ statement gets executed. Second, ‘intNumA’ has not been defined in the function but since the function is at a higher scope it can still access the variable. Defining variables with the same name in multiple scopes is a concept called shadowing and should normally be avoided. Figure 3 below shows the shadowing of ‘intNumA’.

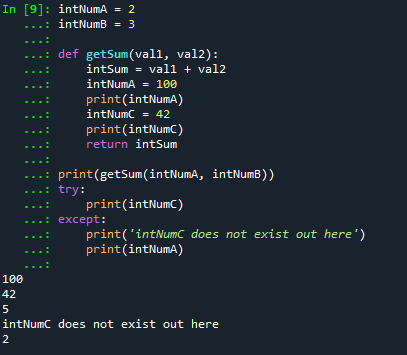


Figure - Shadowing intNumA

While this does work it can lead to confusion. Inside the function ‘getSum’, ‘intNumA’ acts as a local variable. This is why ‘intNumA’ is not changed to ‘100’ in the main body of the code.

Doc String is used to explain how a function works to other programmers and it usually contains five pieces of information. The five parts are: a quick explanation, an in-depth explanation, arguments, returns, and errors. Appendix A.3 contains examples of functions utilizing Doc String.

# Program: CD Inventory with Functions

We were tasked with updating a program that would enable someone to save a digital record of their CDs, load the record from a file, and be able to delete specific saved data. The code for this can be seen in Appendix B.1. The updates required were: create functions to add a CD, create functions to delete a CD, create functions to save the current inventory, organize the coding, and add Doc String.

I split the add CD option into two functions: one for input and output, and another for processing. The first takes three inputs from the user ID, Title, and Artist. I forced the user to only use integers for the ID. I accomplished this by setting up a ‘while’ loop and then using a ‘try’ statement to check if I could convert the input to an integer. The second function uses the return of the first to update the active table. For the delete option, I created a function that expected an ID input from the user. It uses the ID to search the active table and then delete the CD if found. This function returns a Boolean type to track the success of the deletion and the main body uses this to communicate the success to the user. The code required to save the active table to memory was already present in the main body. This allowed me to move it into a function with minimal modification.

# Summary

I was successful in updating the CD Inventory program. I enjoyed learning more about functions and that it is possible to nest them together. The most difficult part this week was separating functions into input/output and processing. It was simpler, for me, to think about a menu option as one thing and then creating a function to do the one thing. This is a concept I will have to practice some more as I continue to learn the programming mind set.

# Appendix A.1

1. #-------------------------------------------#
2. #Tittle: Lab 06-A, IT FDN 110
3. #Desc: Demonstrates Functions concept, based on Basic\_Math.py (Assignment02)
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020-Aug-16, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------
9. intNumA = None
10. intNumB = None
12. #PROCESS------------------------------------
14. **def** getSum(val1, val2):
15. intSum = val1 + val2
16. **return** intSum
18. **def** getDif(val1 = 1, val2 = 1):
19. intDif = intNumA - intNumB
20. **return** intDif
22. **def** getPro(val1, val2):
23. intPro = intNumA \* intNumB
24. **return** intPro
26. **def** getQuo(val1, val2):
27. intQuo = intNumA / intNumB
28. **return** intQuo
30. #PRESENTATION INPUT/OUTPUT------------------
31. # Get User input data
32. **print**('Basic MAth script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
33. intNumA = int(input('Please enter the 1st number: '))
34. intNumB = int(input('Please enter the 2nd number '))
35. # Display the Results
36. **print**('\n\nThis script calculated using the Numbers', intNumA, 'and', intNumB)
37. **print**('The Results are:\n')
38. #In getSum val1 and val2 are given the value passed into the function (intNumA, intNumB) if nothing was passed in we would get an error
39. **print**('Sum:\t\t', getSum(intNumA, intNumB), '\nDifference:\t', getDif())
40. #Note on getDif: I ran into an error whe I did not pass in the arguments when calling getDif()
41. #by assigning values to val1 and val2 I was able to get arround my error.
42. #HOWEVER, this is essentailly a more complicated version of the original code.
43. #Removing val1 = 1, val2 = 1 from the function definition executes as expected
44. **print**('Product:\t', getPro(intNumA, intNumB), '\nQuotuent:\t', getQuo(intNumA, intNumB))

# Appendix A.2

1. #-------------------------------------------#
2. #Tittle: Lab 06-B, IT FDN 110
3. #Desc: Demonstrates Functions concept: One function to rule them all, one function to find them....
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020-Aug-16, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------
9. intNumA = None
10. intNumB = None
12. #PROCESS------------------------------------
14. **def** getMath(val1, val2):
15. intSum = val1 + val2
16. intDif = val1 - val2
17. intPro = intNumA \* intNumB
18. intQuo = intNumA / intNumB
19. **return** intSum, intDif, intPro, intQuo
21. #PRESENTATION INPUT/OUTPUT------------------
22. # Get User input data
23. **print**('Basic MAth script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
24. intNumA = int(input('Please enter the 1st number: '))
25. intNumB = int(input('Please enter the 2nd number '))
26. one\_Answer = getMath(intNumA, intNumB)
27. # Display the Results
28. **print**('\n\nThis script calculated using the Numbers', intNumA, 'and', intNumB)
29. **print**('The Results are:\n')
31. **print**('Sum:\t\t', one\_Answer[0], '\nDifference:\t', one\_Answer[1])
33. **print**('Product:\t', one\_Answer[2], '\nQuotuent:\t', one\_Answer[3])

# Appendix A.3

1. #-------------------------------------------#
2. #Tittle: Lab 06-C, IT FDN 110
3. #Desc: Demonstrates Functions and Class concept, includes doc string
4. #Change Log: (Who, When, What)
5. #Brent Kieszling, 2020-Aug-16, created file
6. #-------------------------------------------#
8. #DATA---------------------------------------
9. intNumA = None
10. intNumB = None
12. #PROCESS------------------------------------
14. **class** SimpleMath():
15. """ Preforms the following operations on the two inputs (+,-,/,\*).
17. This class takes two arguments val1 and val2 and performs four differnt
18. opterations:(1) Addition. (2) Subtraction. (3) Multiplication. (4) Division
20. Args:
21. val1: Is the first integer input
22. val2: Is the second integer input
24. Retuns:
25. (1) Addition.
26. (2) Subtraction.
27. (3) Multiplication.
28. (4) Division.
29. """
30. @staticmethod
31. **def** getSum(val1, val2):
32. """ Adds two numbers.
34. This sums the numbers val1 and val2.
36. Args:
37. val1: Is the first integer input
38. val2: Is the second integer input
40. Retuns:
41. The sum of the inputs
42. """
43. intSum = val1 + val2
44. **return** intSum
45. @staticmethod
46. **def** getDif(val1, val2):
47. """ Subtracts two numbers.
49. This subtracts val2 from val1.
51. Args:
52. val1: Is the first integer input
53. val2: Is the second integer input
55. Retuns:
56. The difference of the inputs
57. """
58. intDif = intNumA - intNumB
59. **return** intDif
60. @staticmethod
61. **def** getPro(val1, val2):
62. """ Multiplies two numbers.
64. This multiplies val1 and val2.
66. Args:
67. val1: Is the first integer input
68. val2: Is the second integer input
70. Retuns:
71. The product of the inputs
72. """
73. intPro = intNumA \* intNumB
74. **return** intPro
75. @staticmethod
76. **def** getQuo(val1, val2):
77. """ Adds two numbers.
79. This determines the quotient of val1 and val2.
81. Args:
82. val1: Is the first integer input
83. val2: Is the second integer input
85. Retuns:
86. The quotient of the inputs
87. """
88. intQuo = intNumA / intNumB
89. **return** intQuo
91. #PRESENTATION INPUT/OUTPUT------------------
92. # Get User input data
93. **print**('Basic MAth script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
94. intNumA = int(input('Please enter the 1st number: '))
95. intNumB = int(input('Please enter the 2nd number '))
96. # Display the Results
97. **print**('\n\nThis script calculated using the Numbers', intNumA, 'and', intNumB)
98. **print**('The Results are:\n')
100. **print**('Sum:\t\t', SimpleMath.getSum(intNumA, intNumB), '\nDifference:\t', SimpleMath.getDif(intNumA, intNumB))

103. **print**('Product:\t', SimpleMath.getPro(intNumA, intNumB), '\nQuotuent:\t', SimpleMath.getQuo(intNumA, intNumB))

# Appendix B.1

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. #Brent Kieszling, 2020-Aug-17, Added function new\_cd and add\_cd
7. #Brent Kieszling, 2020-Aug-18, Updated function write\_file
8. #Brent Kieszling, 2020-Aug-19, Fixed
9. #------------------------------------------#
11. **import** os
13. # -- DATA -- #
14. strChoice = '' # User input
15. lstTbl = []  # list of lists to hold data
16. dicRow = {'ID': '', 'Title': '', 'Artist': ''}  # list of data row
17. strFileName = 'CDInventory.txt'  # data storage file
18. objFile = None  # file object
20. # -- PROCESSING -- #
21. **if** os.path.exists(strFileName) != True:
22. objFile = open(strFileName, 'a')
23. objFile.close()
24. **class** DataProcessor:
25. """Processes the data"""
26. @staticmethod
27. **def** add\_cd(tplNewCD):
28. """Adds a new CD
30. Takes the user input tplNewCD and assigns each value to a dictionary with keys:
31. ID, Title, and Artist. Then it adds it to the active table.
33. Args:
34. tplNewCD (tuple): Contains 3 items: ID, Title, and Artist.
36. Returns:
37. None.
38. """
39. dicRow = {'ID': tplNewCD[0], 'Title': tplNewCD[1], 'Artist': tplNewCD[2]}
40. lstTbl.append(dicRow)
42. @staticmethod
43. **def** delete\_cd(remove\_ID):
44. """Deletes a CD
46. Takes the user input remove\_ID and searches the active table for the
47. appropriate ID and removes it. Additionaly, tracks success
48. via blnCDRemoved.
50. Args:
51. remove\_ID (interger): Holds the ID the user requested be removed.
53. Returns:
54. blnCDRemoved(boolean): True if row removed otherwise False.
55. """
56. intRowNr = -1
57. blnCDRemoved = False
58. **for** row **in** lstTbl:
59. intRowNr += 1
60. **if** row['ID'] == intIDDel:
61. **del** lstTbl[intRowNr]
62. blnCDRemoved = True
63. **break**
64. **return** blnCDRemoved


68. **class** FileProcessor:
69. """Processing the data to and from text file"""
71. @staticmethod
72. **def** read\_file(file\_name, table):
73. """Function to manage data ingestion from file to a list of dictionaries
75. Reads the data from file identified by file\_name into a 2D table
76. (list of dicts) table one line in the file represents one dictionary row in table.
78. Args:
79. file\_name (string): name of file used to read the data from
80. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
82. Returns:
83. None.
84. """
85. objFile = open(file\_name, 'r')
86. **for** line **in** objFile:
87. data = line.strip().split(',')
88. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
89. **if** dicRow **in** table:
90. **pass**
91. **else**:
92. table.append(dicRow)
93. objFile.close()
95. @staticmethod
96. **def** write\_file(file\_name, table):
97. objFile = open(strFileName, 'w')
98. **for** row **in** table:
99. lstValues = list(row.values())
100. lstValues[0] = str(lstValues[0])
101. objFile.write(','.join(lstValues) + '\n')
102. objFile.close()

105. # -- PRESENTATION (Input/Output) -- #
107. **class** IO:
108. """Handling Input / Output"""
110. @staticmethod
111. **def** print\_menu():
112. """Displays a menu of choices to the user
114. Args:
115. None.
117. Returns:
118. None.
119. """
121. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
122. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
124. @staticmethod
125. **def** menu\_choice():
126. """Gets user input for menu selection
128. Args:
129. None.
131. Returns:
132. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
134. """
135. choice = ' '
136. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
137. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
138. **print**()  # Add extra space for layout
139. **return** choice
141. @staticmethod
142. **def** show\_inventory(table):
143. """Displays current inventory table

146. Args:
147. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
149. Returns:
150. None.
152. """
153. **print**('======= The Current Inventory: =======')
154. **print**('ID\tCD Title (by: Artist)\n')
155. **for** row **in** table:
156. **print**('{}\t{} (by:{})'.format(\*row.values()))
157. **print**('======================================')
159. @staticmethod
160. **def** new\_cd():
161. """Allows the user to add a CD to the active inventory table
163. Args:
164. None.
166. Returns:
167. intID (interger): Serialized ID
168. strTitle (string): Tittle of CD
169. stArtist (string): Name of artist
171. """
172. # 3.3.1 Ask user for new ID, CD Title and Artist
173. strID = input('Enter ID: ').strip()
174. **while** True:
175. **try**:
176. intID = int(strID)
177. **break**
178. **except**:
179. strID = input('Please enter an interger for the ID.')
180. strTitle = input('What is the CD\'s title? ').strip()
181. stArtist = input('What is the Artist\'s name? ').strip()
182. **return** intID, strTitle, stArtist

185. # 1. When program starts, read in the currently saved Inventory
187. FileProcessor.read\_file(strFileName, lstTbl)
189. # 2. start main loop
190. **while** True:
191. # 2.1 Display Menu to user and get choice
192. IO.print\_menu()
194. # 3. Process menu selection
195. strChoice = IO.menu\_choice()
196. # 3.1 process exit first
197. **if** strChoice == 'x':
198. **break**
199. # 3.2 process load inventory
200. **if** strChoice == 'l':
201. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
202. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled. ')
203. **if** strYesNo.lower() == 'yes':
204. **print**('reloading...')
205. FileProcessor.read\_file(strFileName, lstTbl)
206. IO.show\_inventory(lstTbl)
207. **else**:
208. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
209. IO.show\_inventory(lstTbl)
210. **continue**  # start loop back at top.
211. # 3.3 process add a CD
212. **elif** strChoice == 'a':
213. #This casts the return from IO.new\_cd() into the function DataProcessor.add\_cd
214. DataProcessor.add\_cd(IO.new\_cd())
215. IO.show\_inventory(lstTbl)
216. **continue**  # start loop back at top.
217. # 3.4 process display current inventory
218. **elif** strChoice == 'i':
219. IO.show\_inventory(lstTbl)
220. **continue**  # start loop back at top.
221. # 3.5 process delete a CD
222. **elif** strChoice == 'd':
223. # 3.5.1 get Userinput for which CD to delete
224. # 3.5.1.1 display Inventory to user
225. IO.show\_inventory(lstTbl)
226. # 3.5.1.2 ask user which ID to remove
227. intIDDel = int(input('Which ID would you like to delete? ').strip())
228. # 3.5.2 search thru table and delete CD
229. blnCDRemoved = DataProcessor.delete\_cd(intIDDel)
230. **if** blnCDRemoved:
231. **print**('The CD was removed')
232. **else**:
233. **print**('Could not find this CD!')
234. IO.show\_inventory(lstTbl)
235. **continue**  # start loop back at top.
236. # 3.6 process save inventory to file
237. **elif** strChoice == 's':
238. # 3.6.1 Display current inventory and ask user for confirmation to save
239. IO.show\_inventory(lstTbl)
240. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
241. # 3.6.2 Process choice
242. **if** strYesNo == 'y':
243. # 3.6.2.1 save data
244. FileProcessor.write\_file(strFileName, lstTbl)
245. **else**:
246. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
247. **continue**  # start loop back at top.
248. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
249. **else**:
250. **print**('General Error')

# Appendix B.2

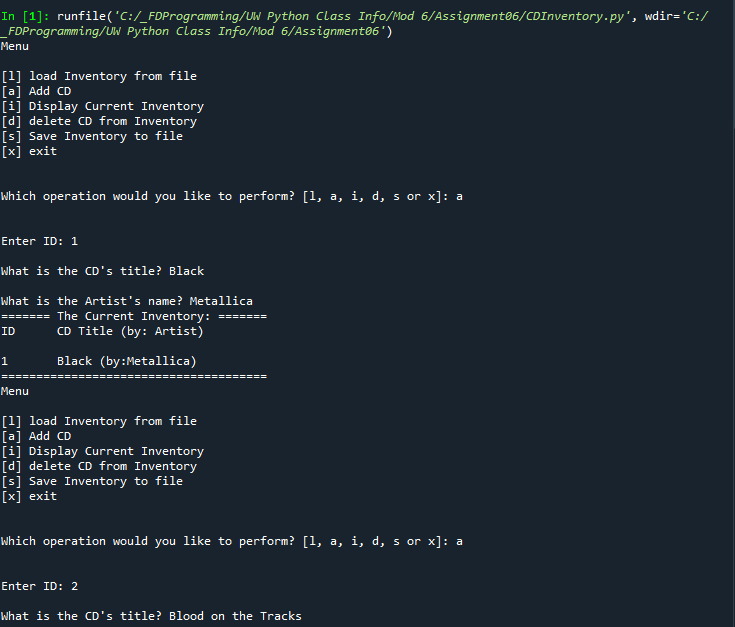


Figure 4 - Program run in Spyder part 1/3

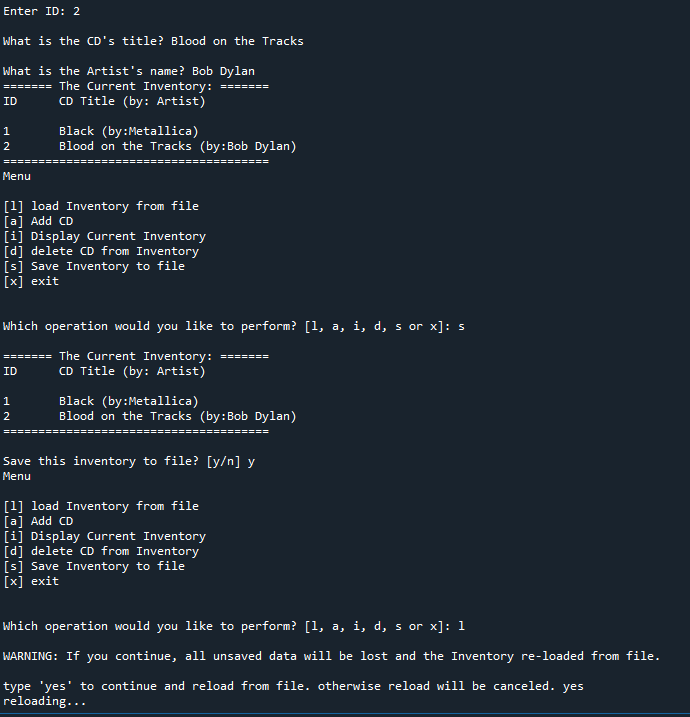


Figure 5 - Program run in Spyder part 2/3

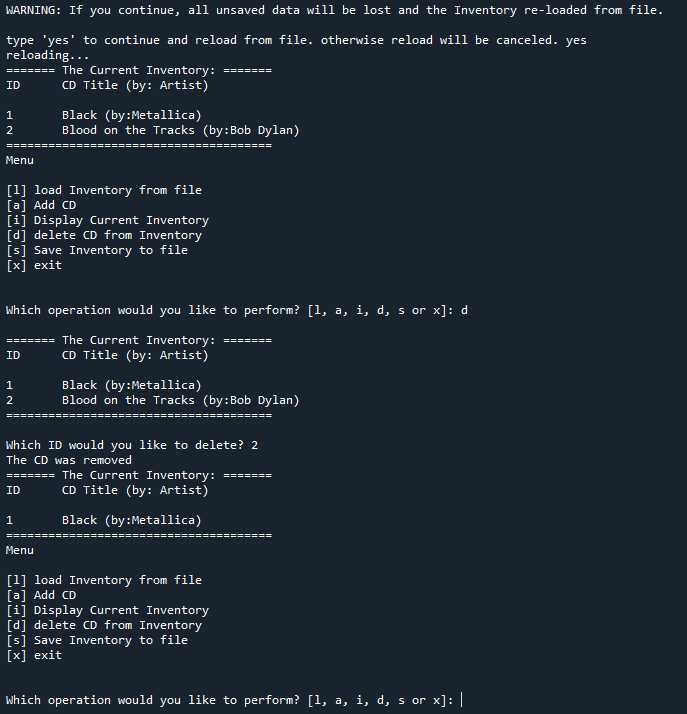


Figure - Program run in Spyder part 3/3

# Appendix B.3

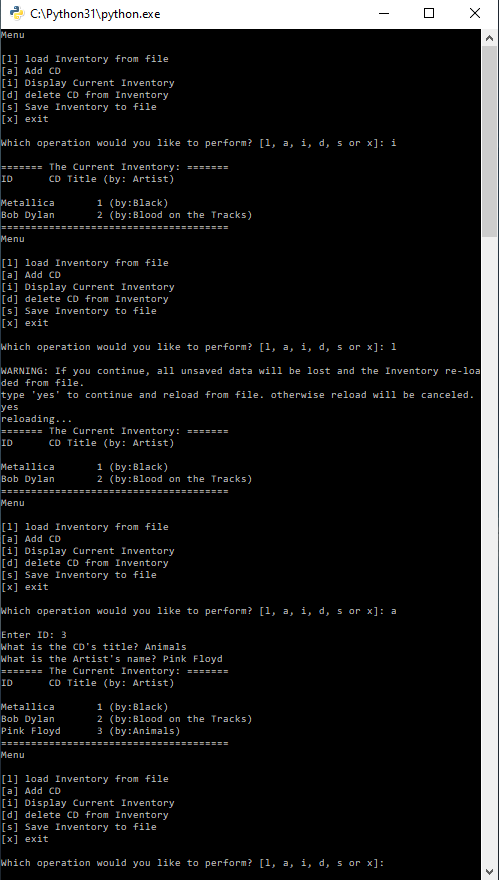


Figure 7 – Program run in Terminal 1

# GitHub Link

1. <https://github.com/Brent-K?tab=repositories>

# References

1. <https://www.w3schools.com/python/python_functions.asp> Accessed: 8/19/2020