<Hongyi Zhu>

<2021-12-12>

<IT FDN 110>

<Assignment\_Module\_09>

Object Oriented Programming: Importing Modules

# Introduction

In this module, we learned to create objects of different classes in the same program and let the objects to communicate with each other. We also used the import command and the main module to make the whole program work.

# Topic 1

I used [Syntax Highlighter](https://highlight.hohli.com/index.php) (external reference)[[1]](#footnote-1) web page to highlight the script as shown in all the following listings.

Listing 1 is the main module CD\_Inventory. And the notes are included as comments and pseudocodes in python files.

1. import ProcessingClasses as PC
2. import IOClasses as IO
4. lstFileNames = ['AlbumInventory.txt', 'TrackInventory.txt']
5. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
7. while True:
8. IO.ScreenIO.print\_menu()
9. strChoice = IO.ScreenIO.menu\_choice()
11. if strChoice == 'x':
12. break
13. if strChoice == 'l':
14. print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
15. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
16. if strYesNo.lower() == 'yes':
17. print('reloading...')
18. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
19. IO.ScreenIO.show\_inventory(lstOfCDObjects)
20. else:
21. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
22. IO.ScreenIO.show\_inventory(lstOfCDObjects)
23. continue  # start loop back at top.
24. elif strChoice == 'a':
25. tplCdInfo = IO.ScreenIO.get\_CD\_info()
26. PC.DataProcessor.add\_CD(tplCdInfo, lstOfCDObjects)
27. IO.ScreenIO.show\_inventory(lstOfCDObjects)
28. continue  # start loop back at top.
29. elif strChoice == 'd':
30. IO.ScreenIO.show\_inventory(lstOfCDObjects)
31. continue  # start loop back at top.
32. elif strChoice == 'c':
33. IO.ScreenIO.show\_inventory(lstOfCDObjects)
34. cd\_idx = input('Select the CD / Album index:')
35. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
36. # TODO add code to handle tracks on an individual CD
37. while True:
38. IO.ScreenIO.print\_CD\_menu()
39. strSubChoice = IO.ScreenIO.menu\_CD\_choice()
40. if strSubChoice == 'x':
41. break
42. if strSubChoice == 'a':
43. trackInfo = IO.ScreenIO.get\_track\_info()
44. PC.DataProcessor.add\_track(trackInfo, cd)
45. continue
46. if strSubChoice == 'd':
47. IO.ScreenIO.show\_tracks(cd)
48. continue
49. if strSubChoice == 'r':
50. IO.ScreenIO.show\_tracks(cd)
51. while True:
52. track\_idx = input('Select the Track index: ')
53. if track\_idx.isdigit():
54. break
55. cd.rmv\_track(int(track\_idx))
56. elif strChoice == 's':
57. IO.ScreenIO.show\_inventory(lstOfCDObjects)
58. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
59. if strYesNo == 'y':
60. IO.FileIO.save\_inventory(lstFileNames, lstOfCDObjects)
61. else:
62. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
63. continue  # start loop back at top.
64. else:
65. print('General Error')

Listing 1 - CDInventory.py

Listing 2 is the python script of Data Classes.

1. if \_\_name\_\_ == '\_\_main\_\_':
2. raise Exception('This file is not meant to run by itself')
4. class Track():
5. # Add Track class code
6. # -- Constructor -- #
7. def \_\_init\_\_(self, position:int, title:str, length:str) -> str:
8. # -- Properties -- #
9. self.\_\_position = position
10. self.\_\_title = title
11. self.\_\_length = length
13. # Track position
14. @property
15. def position(self):
16. return self.\_\_position
17. @position.setter
18. def position(self, value):
19. try:
20. value = int(value)
21. except ValueError:
22. raise Exception("Needs an integer as position")
23. if value < 1:
24. raise Exception("Position must greater than 0")
25. self.\_\_position = value
27. # Track title
28. @property
29. def title(self):
30. return self.\_\_title
31. @title.setter
32. def title(self, value):
33. if type(value) == str:
34. self.\_\_title = value
35. else:
36. raise Exception("Title must a str")
38. # Track length
39. @property
40. def length(self):
41. return self.\_\_length
42. @length.setter
43. def length(self, value):
44. if type(value) == str:
45. self.\_\_length = value
46. else:
47. raise Exception("Length must a str")
49. # -- Methods -- #
50. # Add Track class methods
51. def \_\_str\_\_(self):
52. """Returns Track details as formatted string"""
53. return '{:>2}. {} (len:{})'.format(self.position, self.title, self.length)
55. def get\_record(self) -> str:
56. """Returns: Track record formatted for saving to file"""
57. return '{},{},{}\n'.format(self.position, self.title, self.length)

60. class CD:
61. # Modify CD class as required
62. # -- Constructor -- #
63. def \_\_init\_\_(self, cd\_id: int, cd\_title: str, cd\_artist: str) -> None:
64. #    -- Attributes  -- #
65. try:
66. self.\_\_cd\_id = int(cd\_id)
67. self.\_\_cd\_title = str(cd\_title)
68. self.\_\_cd\_artist = str(cd\_artist)
69. self.\_\_tracks = []
70. except Exception as e:
71. raise Exception('CD initial values are error!\n' + str(e))
73. # -- Properties -- #
74. # CD ID
75. @property
76. def cd\_id(self):
77. return self.\_\_cd\_id
79. @cd\_id.setter
80. def cd\_id(self, value):
81. try:
82. self.\_\_cd\_id = int(value)
83. except Exception:
84. raise Exception('ID needs to be Integer')
86. # CD title
87. @property
88. def cd\_title(self):
89. return self.\_\_cd\_title
91. @cd\_title.setter
92. def cd\_title(self, value):
93. try:
94. self.\_\_cd\_title = str(value)
95. except Exception:
96. raise Exception('Title needs to be String!')
98. # CD artist
99. @property
100. def cd\_artist(self):
101. return self.\_\_cd\_artist
103. @cd\_artist.setter
104. def cd\_artist(self, value):
105. try:
106. self.\_\_cd\_artist = str(value)
107. except Exception:
108. raise Exception('Artist needs to be String!')
110. # CD tracks
111. @property
112. def cd\_tracks(self):
113. return self.\_\_tracks
115. # -- Methods -- #
116. def \_\_str\_\_(self):
117. """Returns: CD details as formatted string"""
118. return '{:>2}\t{} (by: {})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
120. def get\_record(self):
121. """Returns: CD record formatted for saving to file"""
122. return '{},{},{}\n'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
124. def add\_track(self, track: Track) -> None:
125. # Append track
126. self.\_\_tracks.append(track)
127. # Sort tracks
128. self.\_\_sort\_tracks()
130. def rmv\_track(self, track\_id: int) -> None:
131. # Remove track
132. if track\_id > len(self.\_\_tracks):
133. return
134. del self.\_\_tracks[track\_id - 1]
135. # Sort tracks
136. self.\_\_sort\_tracks()
138. def \_\_sort\_tracks(self):
139. n = len(self.\_\_tracks)
140. for track in self.\_\_tracks:
141. if (track is not None) and (n < track.position):
142. n = track.position
143. tmp\_tracks = [None] \* n
144. for track in self.\_\_tracks:
145. if track is not None:
146. tmp\_tracks[track.position - 1] = track
147. self.\_\_tracks = tmp\_tracks
149. def get\_tracks(self) -> str:
150. self.\_\_sort\_tracks()
151. if len(self.\_\_tracks) < 1:
152. raise Exception('No tracks saved for this Album')
153. result = ''
154. for track in self.\_\_tracks:
155. if track is None:
156. result += 'No Information for this track\n'
157. else:
158. result += str(track) + '\n'
159. return result
161. def get\_long\_record(self) -> str:
162. result = self.get\_record() + '\n'
163. result += self.get\_tracks() + '\n'
164. return result

Listing 2 - DataClasses.py

Listing 3 is the python script of IO Classes.

1. if \_\_name\_\_ == '\_\_main\_\_':
2. raise Exception('This file is not meant to run by itself')
4. import DataClasses as DC
5. import ProcessingClasses as PC
7. class FileIO:
8. @staticmethod
9. def save\_inventory(file\_name: list, lst\_Inventory: list) -> None:
10. # Modify method to accept a list of file names.
11. file\_name\_CD = file\_name[0]
12. file\_name\_TRACK = file\_name[1]
13. try:
14. with open(file\_name\_CD, 'w') as file:
15. for disc in lst\_Inventory:
16. file.write(disc.get\_record())
17. # TODO add code to save track data to file
18. with open(file\_name\_TRACK, 'w') as file:
19. for disc in lst\_Inventory:
20. tracks = disc.cd\_tracks
21. cd\_id = disc.cd\_id
22. for track in tracks:
23. if track is not None:
24. record = track.get\_record()
25. row = '{},{}'.format(cd\_id, record)
26. file.write(row)
27. except Exception as e:
28. print('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')
30. @staticmethod
31. def load\_inventory(file\_name: list) -> list:
32. lst\_Inventory = []
33. # Modify method to accept a list of file names
34. file\_name\_CD = file\_name[0]
35. file\_name\_TRACK = file\_name[1]
36. try:
37. with open(file\_name\_CD, 'r') as file:
38. for line in file:
39. data = line.strip().split(',')
40. row = DC.CD(data[0], data[1], data[2])
41. lst\_Inventory.append(row)
42. # TODO add code to load track data
43. with open(file\_name\_TRACK, 'r') as file:
44. for line in file:
45. data = line.strip().split(',')
46. cd\_idx = int(data[0])
47. cd = PC.DataProcessor.select\_cd(lst\_Inventory, cd\_idx)
48. track = DC.Track(int(data[1]), data[2], data[3])
49. cd.add\_track(track)
50. except Exception as e:
51. print('File has errors', e, e.\_\_doc\_\_, sep='\n')
52. return lst\_Inventory
54. class ScreenIO:
55. @staticmethod
56. def print\_menu():
57. print('Main Menu\n\n[l] load Inventory from file\n[a] Add CD / Album\n[d] Display Current Inventory')
58. print('[c] Choose CD / Album\n[s] Save Inventory to file\n[x] exit\n')
60. @staticmethod
61. def menu\_choice():
62. choice = ' '
63. while choice not in ['l', 'a', 'd', 'c', 's', 'x']:
64. choice = input('Which operation would you like to perform? [l, a, d, c, s or x]: ').lower().strip()
65. print()  # Add extra space for layout
66. return choice
68. @staticmethod
69. def print\_CD\_menu():
70. print('CD Sub Menu\n\n[a] Add track\n[d] Display cd / Album details\n[r] Remove track\n[x] exit to Main Menu')
72. @staticmethod
73. def menu\_CD\_choice():
74. choice = ' '
75. while choice not in ['a', 'd', 'r', 'x']:
76. choice = input('Which operation would you like to perform? [a, d, r or x]: ').lower().strip()
77. print()  # Add extra space for layout
78. return choice
80. @staticmethod
81. def show\_inventory(table):
82. print('======= The Current Inventory: =======')
83. print('ID\tCD Title (by: Artist)\n')
84. for row in table:
85. print(row)
86. print('======================================')
88. @staticmethod
89. def show\_tracks(cd):
90. print('====== Current CD / Album: ======')
91. print(cd)
92. print('=================================')
93. print(cd.get\_tracks())
94. print('=================================')
96. @staticmethod
97. def get\_CD\_info():
98. cdId = input('Enter ID: ').strip()
99. cdTitle = input('What is the CD\'s title? ').strip()
100. cdArtist = input('What is the Artist\'s name? ').strip()
101. return cdId, cdTitle, cdArtist
103. @staticmethod
104. def get\_track\_info():
105. trkId = input('Enter Position on CD / Album: ').strip()
106. trkTitle = input('What is the Track\'s title? ').strip()
107. trkLength = input('What is the Track\'s length? ').strip()
108. return trkId, trkTitle, trkLength

Listing 3 - IOClasses.py

Listing 4 is the python script of processing classes.

1. if \_\_name\_\_ == '\_\_main\_\_':
2. raise Exception('This file is not meant to ran by itself')
4. import DataClasses as DC
6. class DataProcessor:
7. @staticmethod
8. def add\_CD(CDInfo, table):
9. cdId, title, artist = CDInfo
10. try:
11. cdId = int(cdId)
12. except:
13. raise Exception('ID must be an Integer!')
14. row = DC.CD(cdId, title, artist)
15. table.append(row)
17. @staticmethod
18. def select\_cd(table: list, cd\_idx: int) -> DC.CD:
19. # Add code as required
20. try:
21. cd\_idx = int(cd\_idx)
22. for row in table:
23. if row.cd\_id == cd\_idx:
24. return row
25. except ValueError:
26. print("Needs an Integer as ID")
27. raise ('The '+ str(cd\_idx) + ' CD/Album does not exist')
29. @staticmethod
30. def add\_track(track\_info: tuple, cd: DC.CD) -> None:
31. # Add code as required
32. trackPosition = track\_info[0]
33. trackTitle = track\_info[1]
34. trackLength = track\_info[2]
35. try:
36. trackPosition = int(trackPosition)
37. except:
38. raise Exception('Position must be an Integer')
39. track = DC.Track(trackPosition, trackTitle, trackLength)
40. cd.add\_track(track)

Listing 4 - ProcessingClasses.py

Listing 5 is the python script of testing harness.

1. import DataClasses as DC
2. import ProcessingClasses as PC
3. import IOClasses as IO
5. lstOfCDObjects = []
6. file\_name = ['TestCD.txt', 'TestTrack.txt']
8. print('\n\nTesting Track class')
9. print(DC.Track.\_\_doc\_\_)
10. trk1 = DC.Track(1, 'test.track1', '01:59')
11. trk2 = DC.Track(2, 'test.track2', '02:59')
12. print(trk1)
13. print('record for file:', trk1.get\_record())
15. print('\n\nTesting CD class')
16. print(DC.CD.\_\_doc\_\_)
17. cd1 = DC.CD(1, 'test\_title', 'cd\_artist')
18. print(cd1)
19. print('record for file:', cd1.get\_record())
20. print('adding tracks...')
21. cd1.add\_track(trk1)
22. cd1.add\_track(trk2)
23. print('get tracks:\n', cd1.get\_tracks())
24. print('get long record:\n', cd1.get\_long\_record())
25. print('removing track 2...')
26. cd1.rmv\_track(2)
27. print('get long record:\n', cd1.get\_long\_record())
28. lstOfCDObjects.append(cd1)
30. print('\n\nTesting of class FileIO')
31. IO.FileIO.save\_inventory(file\_name, lstOfCDObjects)
32. print(IO.FileIO.load\_inventory(file\_name))
34. print('\n\nTesting ScreenIO class')
35. print('Main menu:')
36. IO.ScreenIO.print\_menu()
37. print('selection in menu: {}'.format(IO.ScreenIO.menu\_choice()))
38. print('Inventory:')
39. IO.ScreenIO.show\_inventory(lstOfCDObjects)
40. cd2 = DC.CD(2, 'test\_title\_2', 'cd\_artist\_2')
41. lstOfCDObjects.append(cd2)
42. print('Inventory:')
43. for item in lstOfCDObjects:
44. print(item)
45. cd\_idx = 1
46. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
47. print('\nSub Menu')
48. IO.ScreenIO.print\_CD\_menu()
49. print('selection in sub menu: {}'.format(IO.ScreenIO.menu\_CD\_choice()))
50. print('Tracks:')
51. IO.ScreenIO.show\_tracks(cd)
53. print('\n\nTesting Processing Classes')
54. PC.DataProcessor.add\_CD((3, 'Foreigner', 'Foreigner'), lstOfCDObjects)
55. print('Inventory:')
56. for item in lstOfCDObjects:
57. print(item)

Listing 5 - TestHarness.py

# Topic 2

In this topic, I tested TestHarness.py in Spyder and the terminal.

Figure 1 and 2 are the testing results in Spyder.

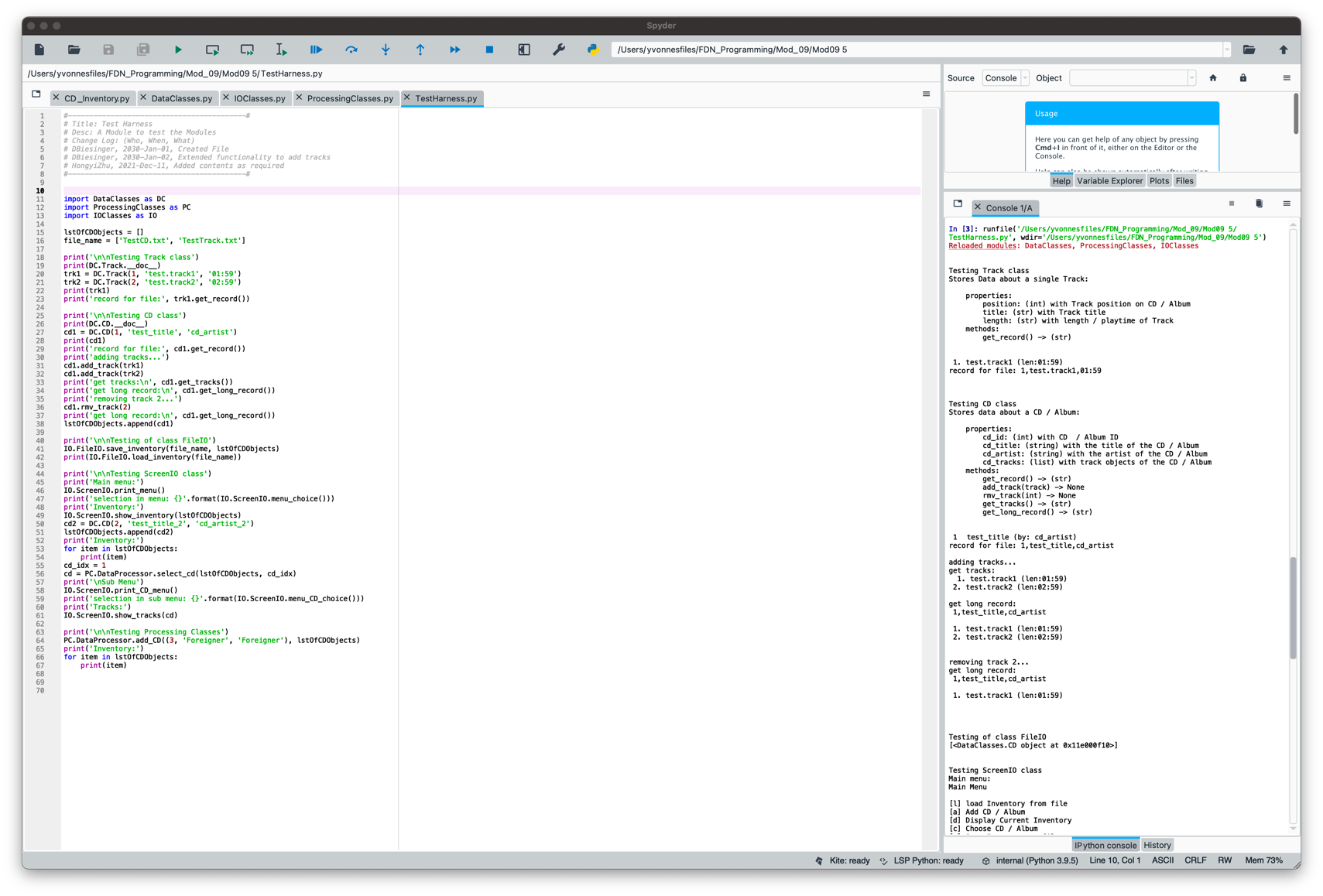


Figure 1 - Running Results of TestHarness.py in Spyder

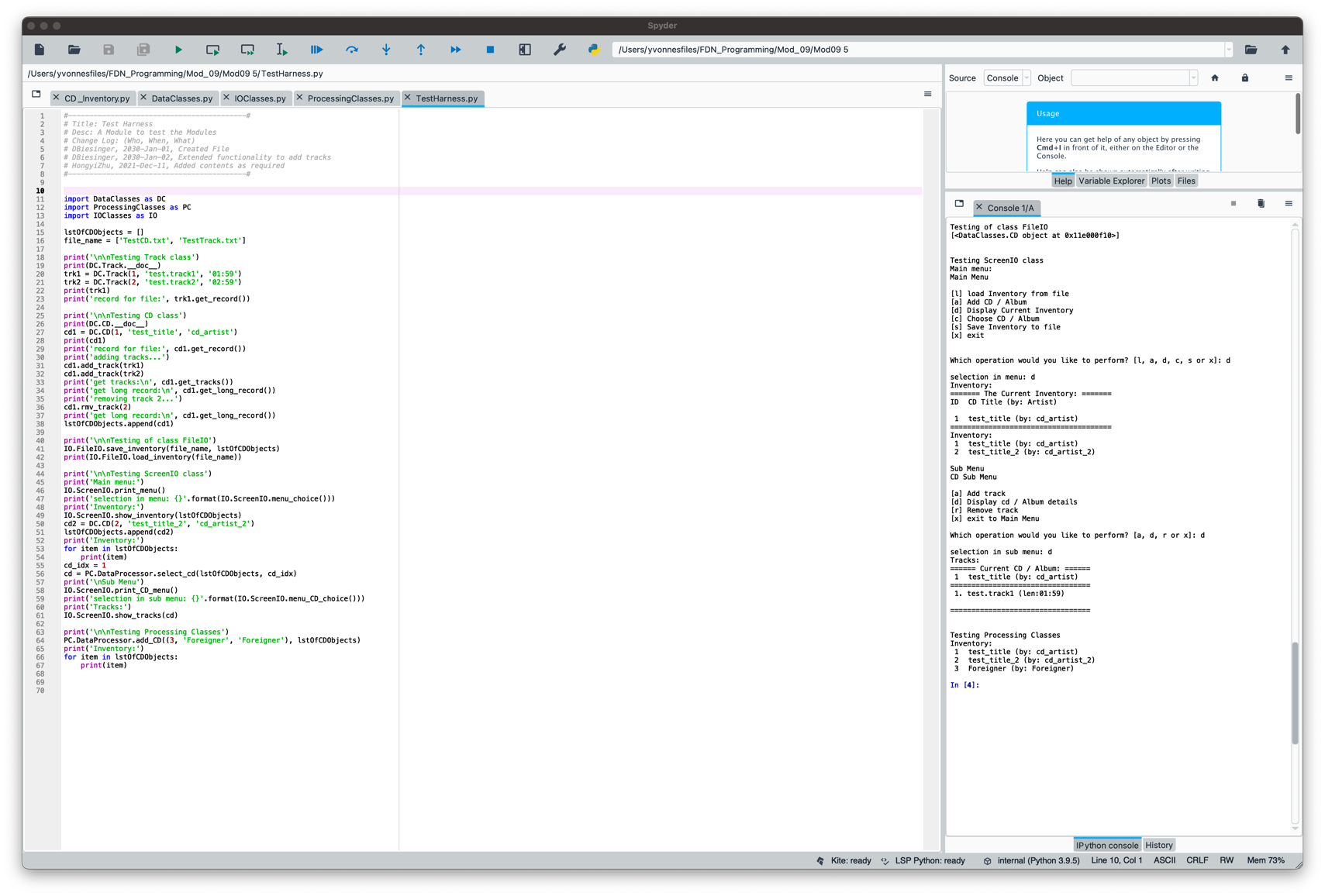


Figure 2 - Running Results of TestHarness.py in Spyder (Continue)

Figure 3 and 4 are the testing results in Terminal.



Figure 3 - Running Results of TestHarness.py in Terminal



Figure 4 - Running Results of TestHarness.py in Terminal (Continue)

# Topic 3

In this topic, I tested CDInventory.py in Spyder and the terminal.

Figure 5 to 7 are the testing results in Spyder.

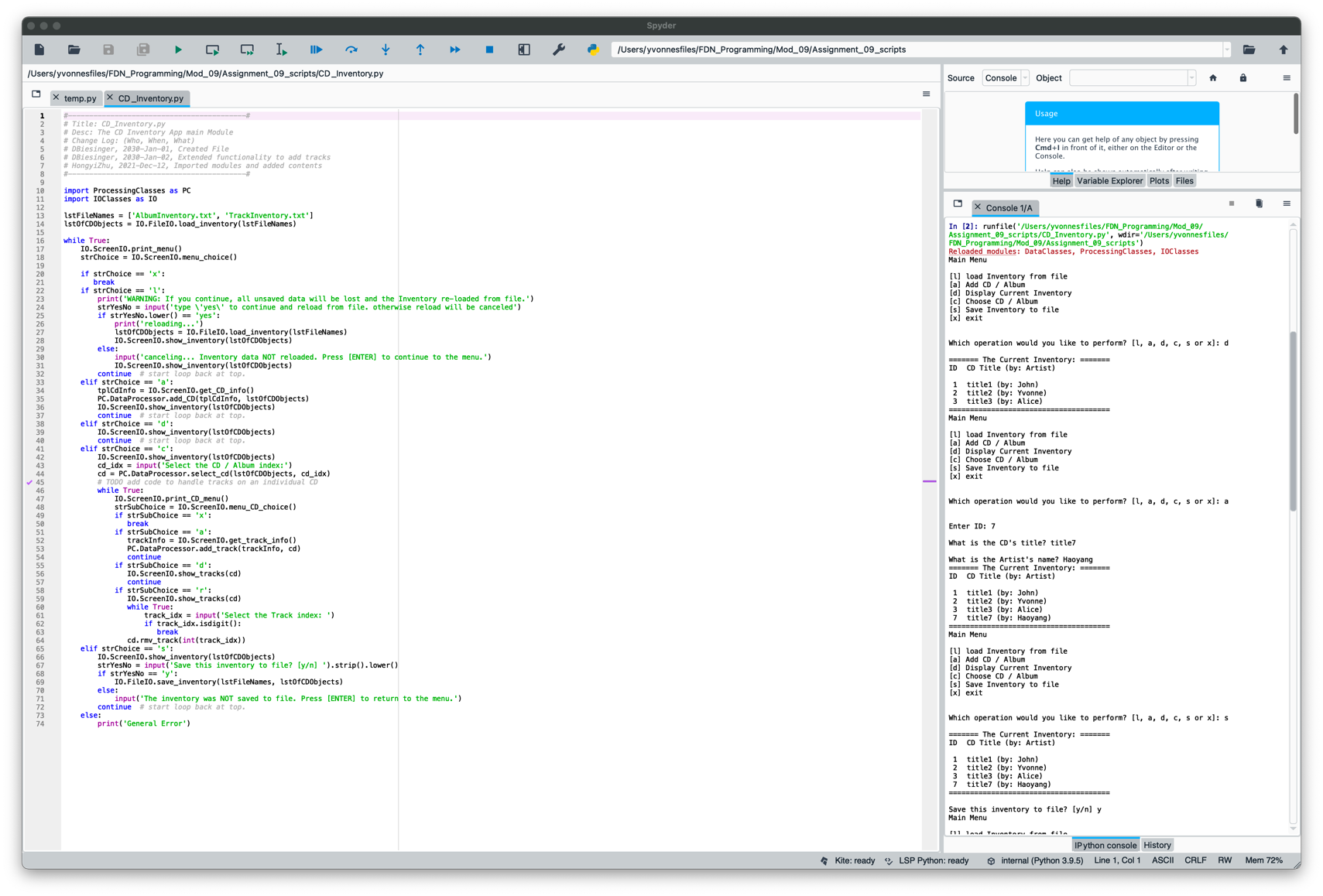


Figure 5 - Running Results of CDInventory.py in Spyder

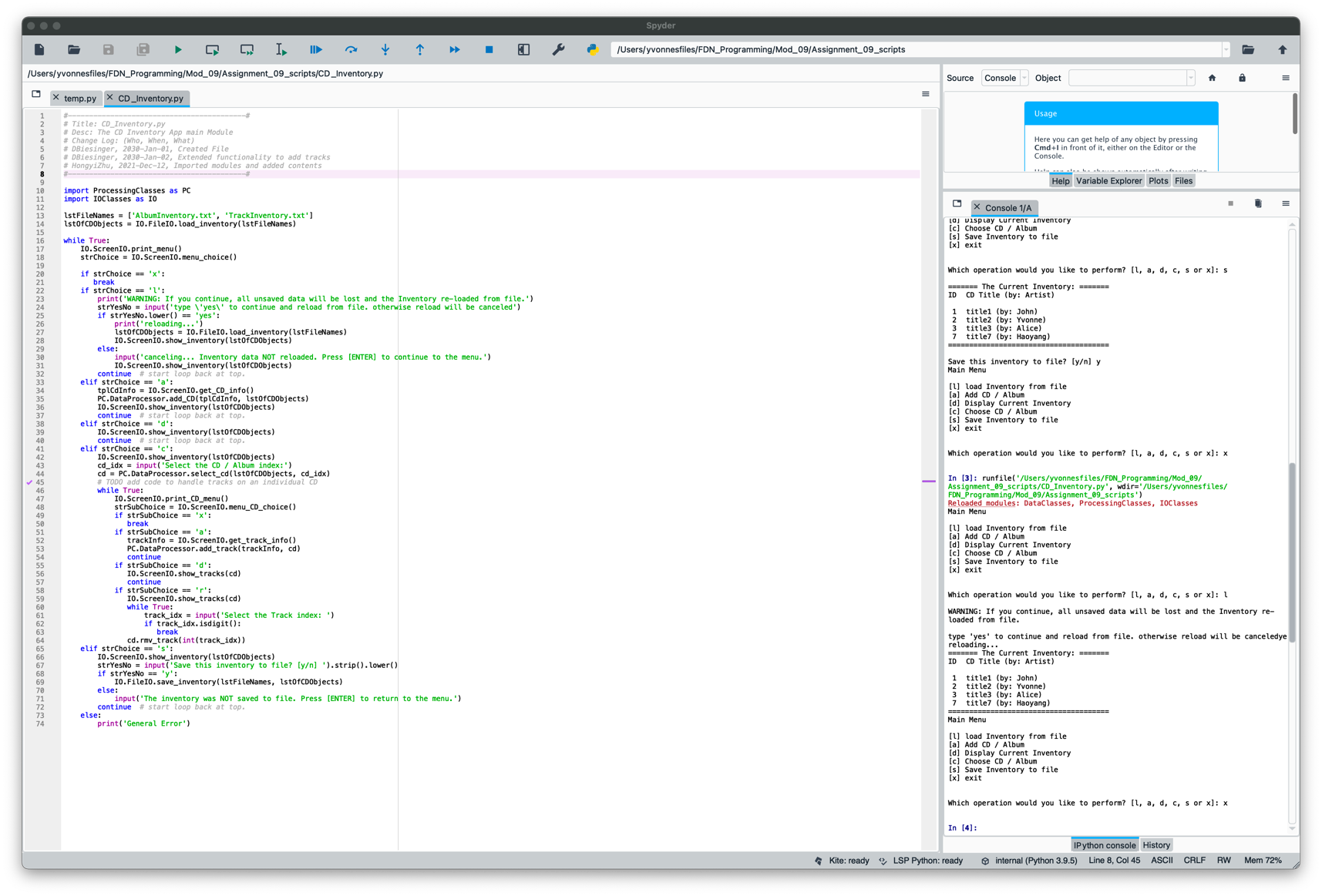


Figure 6 - Running Results of CDInventory.py in Spyder (Continue)

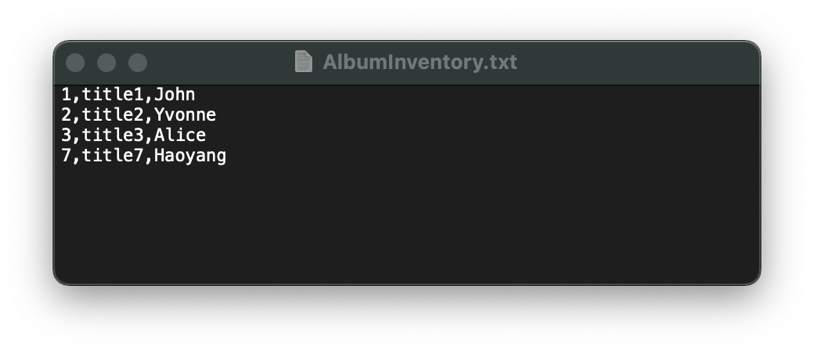


Figure 7 - Corresponding txt Result After Modification in Spyder

Figure 8 to 10 are the testing results in Terminal.

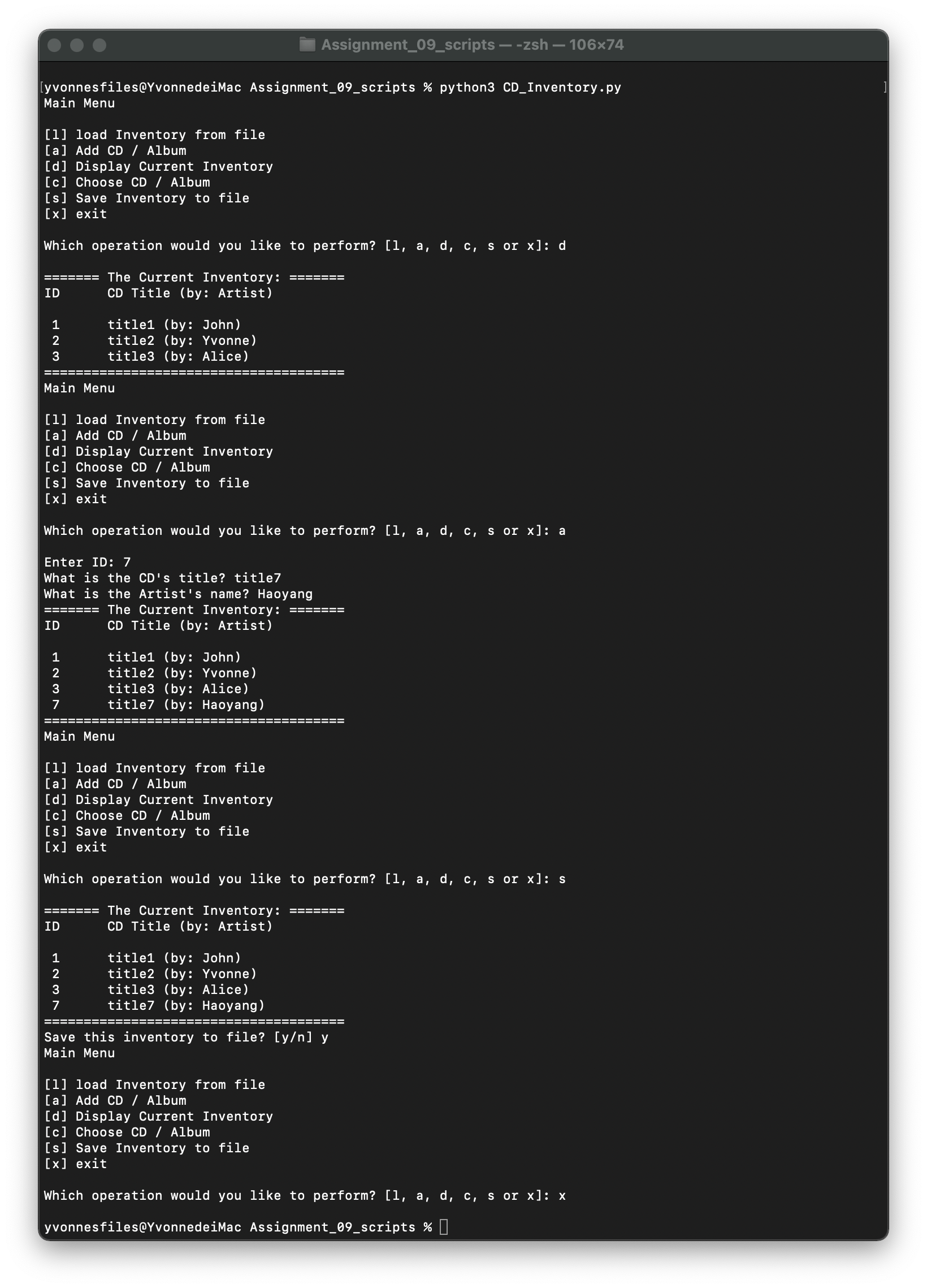


Figure 8 - Running Results of CDInventory.py in Terminal

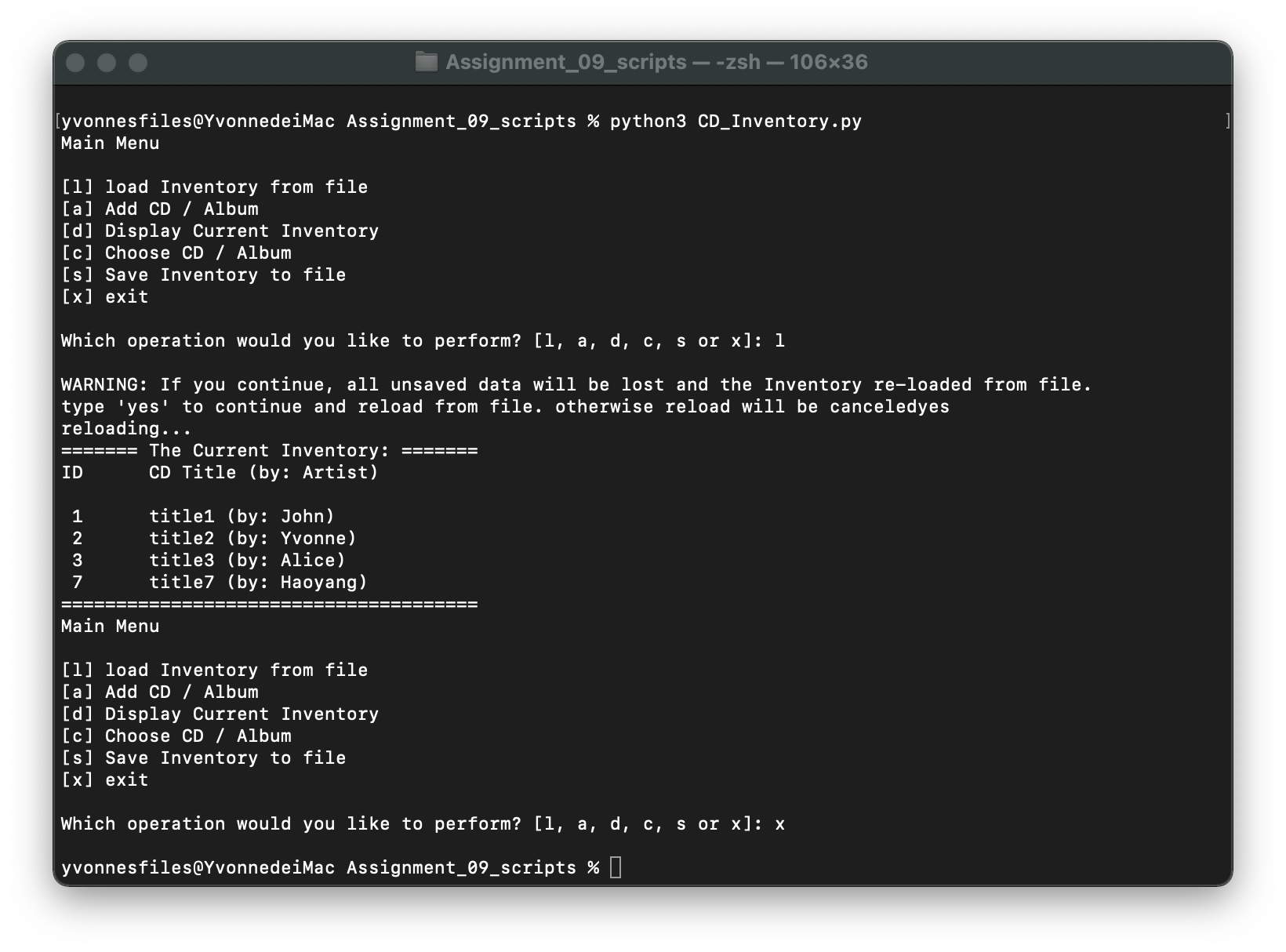


Figure 9 - Running Results of CDInventory.py in Terminal (Continue)

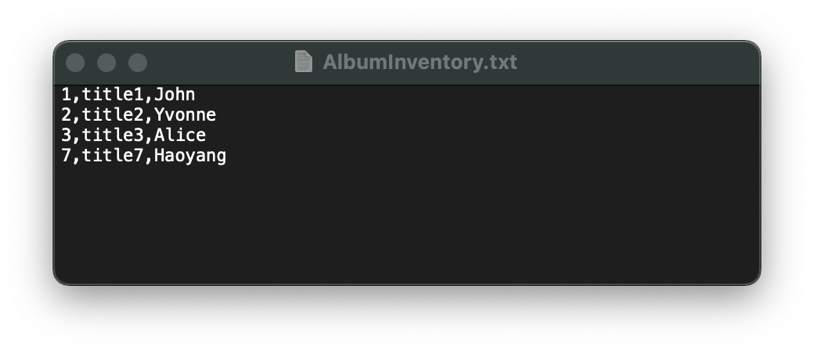


Figure 10 - Corresponding txt Result After Modification in Terminal

# Summary

In this assignment, I created objects of different classes in the same program and allowed objects to communicate with each other. I also created some more complex objects by combining simpler ones and applied them.

# Appendix

# Listing CDInventory.py

1. #------------------------------------------#
2. # Title: CD\_Inventory.py
3. # Desc: The CD Inventory App main Module
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
7. # HongyiZhu, 2021-Dec-12, Imported modules and added contents
8. #------------------------------------------#
10. import ProcessingClasses as PC
11. import IOClasses as IO
13. lstFileNames = ['AlbumInventory.txt', 'TrackInventory.txt']
14. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
16. while True:
17. IO.ScreenIO.print\_menu()
18. strChoice = IO.ScreenIO.menu\_choice()
20. if strChoice == 'x':
21. break
22. if strChoice == 'l':
23. print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
24. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
25. if strYesNo.lower() == 'yes':
26. print('reloading...')
27. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
28. IO.ScreenIO.show\_inventory(lstOfCDObjects)
29. else:
30. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
31. IO.ScreenIO.show\_inventory(lstOfCDObjects)
32. continue  # start loop back at top.
33. elif strChoice == 'a':
34. tplCdInfo = IO.ScreenIO.get\_CD\_info()
35. PC.DataProcessor.add\_CD(tplCdInfo, lstOfCDObjects)
36. IO.ScreenIO.show\_inventory(lstOfCDObjects)
37. continue  # start loop back at top.
38. elif strChoice == 'd':
39. IO.ScreenIO.show\_inventory(lstOfCDObjects)
40. continue  # start loop back at top.
41. elif strChoice == 'c':
42. IO.ScreenIO.show\_inventory(lstOfCDObjects)
43. cd\_idx = input('Select the CD / Album index:')
44. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
45. # TODO add code to handle tracks on an individual CD
46. while True:
47. IO.ScreenIO.print\_CD\_menu()
48. strSubChoice = IO.ScreenIO.menu\_CD\_choice()
49. if strSubChoice == 'x':
50. break
51. if strSubChoice == 'a':
52. trackInfo = IO.ScreenIO.get\_track\_info()
53. PC.DataProcessor.add\_track(trackInfo, cd)
54. continue
55. if strSubChoice == 'd':
56. IO.ScreenIO.show\_tracks(cd)
57. continue
58. if strSubChoice == 'r':
59. IO.ScreenIO.show\_tracks(cd)
60. while True:
61. track\_idx = input('Select the Track index: ')
62. if track\_idx.isdigit():
63. break
64. cd.rmv\_track(int(track\_idx))
65. elif strChoice == 's':
66. IO.ScreenIO.show\_inventory(lstOfCDObjects)
67. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
68. if strYesNo == 'y':
69. IO.FileIO.save\_inventory(lstFileNames, lstOfCDObjects)
70. else:
71. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
72. continue  # start loop back at top.
73. else:
74. print('General Error')

# Listing DataClasses.py

1. #------------------------------------------#
2. # Title: Data Classes
3. # Desc: A Module for Data Classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # DBiesinger, 2030-Jan-02, Modified to add Track class, added methods to CD class to handle tracks
7. # HongyiZhu, 2021-Dec-11, Added contents as required
8. #------------------------------------------#
10. if \_\_name\_\_ == '\_\_main\_\_':
11. raise Exception('This file is not meant to run by itself')
13. class Track():
14. """Stores Data about a single Track:
16. properties:
17. position: (int) with Track position on CD / Album
18. title: (str) with Track title
19. length: (str) with length / playtime of Track
20. methods:
21. get\_record() -> (str)
23. """
24. # Add Track class code
25. # -- Constructor -- #
26. def \_\_init\_\_(self, position:int, title:str, length:str) -> str:
27. # -- Properties -- #
28. self.\_\_position = position
29. self.\_\_title = title
30. self.\_\_length = length
32. # Track position
33. @property
34. def position(self):
35. return self.\_\_position
36. @position.setter
37. def position(self, value):
38. try:
39. value = int(value)
40. except ValueError:
41. raise Exception("Needs an integer as position")
42. if value < 1:
43. raise Exception("Position must greater than 0")
44. self.\_\_position = value
46. # Track title
47. @property
48. def title(self):
49. return self.\_\_title
50. @title.setter
51. def title(self, value):
52. if type(value) == str:
53. self.\_\_title = value
54. else:
55. raise Exception("Title must a str")
57. # Track length
58. @property
59. def length(self):
60. return self.\_\_length
61. @length.setter
62. def length(self, value):
63. if type(value) == str:
64. self.\_\_length = value
65. else:
66. raise Exception("Length must a str")
68. # -- Methods -- #
69. # Add Track class methods
70. def \_\_str\_\_(self):
71. """Returns Track details as formatted string"""
72. return '{:>2}. {} (len:{})'.format(self.position, self.title, self.length)
74. def get\_record(self) -> str:
75. """Returns: Track record formatted for saving to file"""
76. return '{},{},{}\n'.format(self.position, self.title, self.length)

79. class CD:
80. """Stores data about a CD / Album:
82. properties:
83. cd\_id: (int) with CD  / Album ID
84. cd\_title: (string) with the title of the CD / Album
85. cd\_artist: (string) with the artist of the CD / Album
86. cd\_tracks: (list) with track objects of the CD / Album
87. methods:
88. get\_record() -> (str)
89. add\_track(track) -> None
90. rmv\_track(int) -> None
91. get\_tracks() -> (str)
92. get\_long\_record() -> (str)
94. """
95. # Modify CD class as required
96. # -- Constructor -- #
97. def \_\_init\_\_(self, cd\_id: int, cd\_title: str, cd\_artist: str) -> None:
98. #    -- Attributes  -- #
99. try:
100. self.\_\_cd\_id = int(cd\_id)
101. self.\_\_cd\_title = str(cd\_title)
102. self.\_\_cd\_artist = str(cd\_artist)
103. self.\_\_tracks = []
104. except Exception as e:
105. raise Exception('CD initial values are error!\n' + str(e))
107. # -- Properties -- #
108. # CD ID
109. @property
110. def cd\_id(self):
111. return self.\_\_cd\_id
113. @cd\_id.setter
114. def cd\_id(self, value):
115. try:
116. self.\_\_cd\_id = int(value)
117. except Exception:
118. raise Exception('ID needs to be Integer')
120. # CD title
121. @property
122. def cd\_title(self):
123. return self.\_\_cd\_title
125. @cd\_title.setter
126. def cd\_title(self, value):
127. try:
128. self.\_\_cd\_title = str(value)
129. except Exception:
130. raise Exception('Title needs to be String!')
132. # CD artist
133. @property
134. def cd\_artist(self):
135. return self.\_\_cd\_artist
137. @cd\_artist.setter
138. def cd\_artist(self, value):
139. try:
140. self.\_\_cd\_artist = str(value)
141. except Exception:
142. raise Exception('Artist needs to be String!')
144. # CD tracks
145. @property
146. def cd\_tracks(self):
147. return self.\_\_tracks
149. # -- Methods -- #
150. def \_\_str\_\_(self):
151. """Returns: CD details as formatted string"""
152. return '{:>2}\t{} (by: {})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
154. def get\_record(self):
155. """Returns: CD record formatted for saving to file"""
156. return '{},{},{}\n'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
158. def add\_track(self, track: Track) -> None:
159. """Adds a track to the CD / Album

162. Args:
163. track (Track): Track object to be added to CD / Album.
165. Returns:
166. None.
168. """
169. # Append track
170. self.\_\_tracks.append(track)
171. # Sort tracks
172. self.\_\_sort\_tracks()
174. def rmv\_track(self, track\_id: int) -> None:
175. """Removes the track identified by track\_id from Album

178. Args:
179. track\_id (int): ID of track to be removed.
181. Returns:
182. None.
184. """
185. # Remove track
186. if track\_id > len(self.\_\_tracks):
187. return
188. del self.\_\_tracks[track\_id - 1]
189. # Sort tracks
190. self.\_\_sort\_tracks()
192. def \_\_sort\_tracks(self):
193. """Sorts the tracks using Track.position. Fills blanks with None"""
194. n = len(self.\_\_tracks)
195. for track in self.\_\_tracks:
196. if (track is not None) and (n < track.position):
197. n = track.position
198. tmp\_tracks = [None] \* n
199. for track in self.\_\_tracks:
200. if track is not None:
201. tmp\_tracks[track.position - 1] = track
202. self.\_\_tracks = tmp\_tracks
204. def get\_tracks(self) -> str:
205. """Returns a string list of the tracks saved for the Album
207. Raises:
208. Exception: If no tracks are saved with album.
210. Returns:
211. result (string):formatted string of tracks.
213. """
214. self.\_\_sort\_tracks()
215. if len(self.\_\_tracks) < 1:
216. raise Exception('No tracks saved for this Album')
217. result = ''
218. for track in self.\_\_tracks:
219. if track is None:
220. result += 'No Information for this track\n'
221. else:
222. result += str(track) + '\n'
223. return result
225. def get\_long\_record(self) -> str:
226. """gets a formatted long record of the Album: Album information plus track details

229. Returns:
230. result (string): Formatted information about ablum and its tracks.
232. """
233. result = self.get\_record() + '\n'
234. result += self.get\_tracks() + '\n'
235. return result

# Listing IOClasses.py

1. #------------------------------------------#
2. # Title: IO Classes
3. # Desc: A Module for IO Classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
7. # HongyiZhu, 2021-Dec-11, Added contents as required
8. #------------------------------------------#
10. if \_\_name\_\_ == '\_\_main\_\_':
11. raise Exception('This file is not meant to run by itself')
13. import DataClasses as DC
14. import ProcessingClasses as PC
16. class FileIO:
17. """Processes data to and from file:
19. properties:
21. methods:
22. save\_inventory(file\_name, lst\_Inventory): -> None
23. load\_inventory(file\_name): -> (a list of CD objects)
25. """
26. @staticmethod
27. def save\_inventory(file\_name: list, lst\_Inventory: list) -> None:
28. """

31. Args:
32. file\_name (list): list of file names [CD Inventory, Track Inventory] that hold the data.
33. lst\_Inventory (list): list of CD objects.
35. Returns:
36. None.
38. """
40. # Modify method to accept a list of file names.
41. file\_name\_CD = file\_name[0]
42. file\_name\_TRACK = file\_name[1]
43. try:
44. with open(file\_name\_CD, 'w') as file:
45. for disc in lst\_Inventory:
46. file.write(disc.get\_record())
47. # TODO add code to save track data to file
48. with open(file\_name\_TRACK, 'w') as file:
49. for disc in lst\_Inventory:
50. tracks = disc.cd\_tracks
51. cd\_id = disc.cd\_id
52. for track in tracks:
53. if track is not None:
54. record = track.get\_record()
55. row = '{},{}'.format(cd\_id, record)
56. file.write(row)
57. except Exception as e:
58. print('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')
60. @staticmethod
61. def load\_inventory(file\_name: list) -> list:
62. """

65. Args:
66. file\_name (list): list of file names [CD Inventory, Track Inventory] that hold the data.
68. Returns:
69. list: list of CD objects.
71. """
73. lst\_Inventory = []
74. # Modify method to accept a list of file names
75. file\_name\_CD = file\_name[0]
76. file\_name\_TRACK = file\_name[1]
77. try:
78. with open(file\_name\_CD, 'r') as file:
79. for line in file:
80. data = line.strip().split(',')
81. row = DC.CD(data[0], data[1], data[2])
82. lst\_Inventory.append(row)
83. # TODO add code to load track data
84. with open(file\_name\_TRACK, 'r') as file:
85. for line in file:
86. data = line.strip().split(',')
87. cd\_idx = int(data[0])
88. cd = PC.DataProcessor.select\_cd(lst\_Inventory, cd\_idx)
89. track = DC.Track(int(data[1]), data[2], data[3])
90. cd.add\_track(track)
91. except Exception as e:
92. print('File has errors', e, e.\_\_doc\_\_, sep='\n')
93. return lst\_Inventory
95. class ScreenIO:
96. """Handling Input / Output"""
98. @staticmethod
99. def print\_menu():
100. """Displays a menu of choices to the user
102. Args:
103. None.
105. Returns:
106. None.
107. """
109. print('Main Menu\n\n[l] load Inventory from file\n[a] Add CD / Album\n[d] Display Current Inventory')
110. print('[c] Choose CD / Album\n[s] Save Inventory to file\n[x] exit\n')
112. @staticmethod
113. def menu\_choice():
114. """Gets user input for menu selection
116. Args:
117. None.
119. Returns:
120. choice (string): a lower case sting of the users input out of the choices l, a, d, c, s or x
122. """
123. choice = ' '
124. while choice not in ['l', 'a', 'd', 'c', 's', 'x']:
125. choice = input('Which operation would you like to perform? [l, a, d, c, s or x]: ').lower().strip()
126. print()  # Add extra space for layout
127. return choice
129. @staticmethod
130. def print\_CD\_menu():
131. """Displays a sub menu of choices for CD / Album to the user
133. Args:
134. None.
136. Returns:
137. None.
138. """
140. print('CD Sub Menu\n\n[a] Add track\n[d] Display cd / Album details\n[r] Remove track\n[x] exit to Main Menu')
142. @staticmethod
143. def menu\_CD\_choice():
144. """Gets user input for CD sub menu selection
146. Args:
147. None.
149. Returns:
150. choice (string): a lower case sting of the users input out of the choices a, d, r or x
152. """
153. choice = ' '
154. while choice not in ['a', 'd', 'r', 'x']:
155. choice = input('Which operation would you like to perform? [a, d, r or x]: ').lower().strip()
156. print()  # Add extra space for layout
157. return choice
159. @staticmethod
160. def show\_inventory(table):
161. """Displays current inventory table

164. Args:
165. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
167. Returns:
168. None.
170. """
171. print('======= The Current Inventory: =======')
172. print('ID\tCD Title (by: Artist)\n')
173. for row in table:
174. print(row)
175. print('======================================')
177. @staticmethod
178. def show\_tracks(cd):
179. """Displays the Tracks on a CD / Album
181. Args:
182. cd (CD): CD object.
184. Returns:
185. None.
187. """
188. print('====== Current CD / Album: ======')
189. print(cd)
190. print('=================================')
191. print(cd.get\_tracks())
192. print('=================================')
194. @staticmethod
195. def get\_CD\_info():
196. """function to request CD information from User to add CD to inventory

199. Returns:
200. cdId (string): Holds the ID of the CD dataset.
201. cdTitle (string): Holds the title of the CD.
202. cdArtist (string): Holds the artist of the CD.
204. """
206. cdId = input('Enter ID: ').strip()
207. cdTitle = input('What is the CD\'s title? ').strip()
208. cdArtist = input('What is the Artist\'s name? ').strip()
209. return cdId, cdTitle, cdArtist
211. @staticmethod
212. def get\_track\_info():
213. """function to request Track information from User to add Track to CD / Album

216. Returns:
217. trkId (string): Holds the ID of the Track dataset.
218. trkTitle (string): Holds the title of the Track.
219. trkLength (string): Holds the length (time) of the Track.
221. """
223. trkId = input('Enter Position on CD / Album: ').strip()
224. trkTitle = input('What is the Track\'s title? ').strip()
225. trkLength = input('What is the Track\'s length? ').strip()
226. return trkId, trkTitle, trkLength

# Listing ProcessingClasses.py

1. #------------------------------------------#
2. # Title: Processing Classes
3. # Desc: A Module for processing Classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
7. # HongyiZhu, 2021-Dec-11, Added contents as required
8. #------------------------------------------#
10. if \_\_name\_\_ == '\_\_main\_\_':
11. raise Exception('This file is not meant to ran by itself')
13. import DataClasses as DC
15. class DataProcessor:
16. """Processing the data in the application"""
17. @staticmethod
18. def add\_CD(CDInfo, table):
19. """function to add CD info in CDinfo to the inventory table.

22. Args:
23. CDInfo (tuple): Holds information (ID, CD Title, CD Artist) to be added to inventory.
24. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
26. Returns:
27. None.
29. """
31. cdId, title, artist = CDInfo
32. try:
33. cdId = int(cdId)
34. except:
35. raise Exception('ID must be an Integer!')
36. row = DC.CD(cdId, title, artist)
37. table.append(row)
39. @staticmethod
40. def select\_cd(table: list, cd\_idx: int) -> DC.CD:
41. """selects a CD object out of table that has the ID cd\_idx
43. Args:
44. table (list): Inventory list of CD objects.
45. cd\_idx (int): id of CD object to return
47. Raises:
48. Exception: If id is not in list.
50. Returns:
51. row (DC.CD): CD object that matches cd\_idx
53. """
54. # Add code as required
55. try:
56. cd\_idx = int(cd\_idx)
57. for row in table:
58. if row.cd\_id == cd\_idx:
59. return row
60. except ValueError:
61. print("Needs an Integer as ID")
62. raise ('The '+ str(cd\_idx) + ' CD/Album does not exist')
64. @staticmethod
65. def add\_track(track\_info: tuple, cd: DC.CD) -> None:
66. """adds a Track object with attributes in track\_info to cd

69. Args:
70. track\_info (tuple): Tuple containing track info (position, title, Length).
71. cd (DC.CD): cd object the tarck gets added to.
73. Raises:
74. Exception: DESCraised in case position is not an integer.
76. Returns:
77. None: DESCRIPTION.
79. """
81. # Add code as required
82. trackPosition = track\_info[0]
83. trackTitle = track\_info[1]
84. trackLength = track\_info[2]
85. try:
86. trackPosition = int(trackPosition)
87. except:
88. raise Exception('Position must be an Integer')
89. track = DC.Track(trackPosition, trackTitle, trackLength)
90. cd.add\_track(track)

# Listing TestHarness.py

1. #------------------------------------------#
2. # Title: Test Harness
3. # Desc: A Module to test the Modules
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
7. # HongyiZhu, 2021-Dec-11, Added contents as required
8. #------------------------------------------#

11. import DataClasses as DC
12. import ProcessingClasses as PC
13. import IOClasses as IO
15. lstOfCDObjects = []
16. file\_name = ['TestCD.txt', 'TestTrack.txt']
18. print('\n\nTesting Track class')
19. print(DC.Track.\_\_doc\_\_)
20. trk1 = DC.Track(1, 'test.track1', '01:59')
21. trk2 = DC.Track(2, 'test.track2', '02:59')
22. print(trk1)
23. print('record for file:', trk1.get\_record())
25. print('\n\nTesting CD class')
26. print(DC.CD.\_\_doc\_\_)
27. cd1 = DC.CD(1, 'test\_title', 'cd\_artist')
28. print(cd1)
29. print('record for file:', cd1.get\_record())
30. print('adding tracks...')
31. cd1.add\_track(trk1)
32. cd1.add\_track(trk2)
33. print('get tracks:\n', cd1.get\_tracks())
34. print('get long record:\n', cd1.get\_long\_record())
35. print('removing track 2...')
36. cd1.rmv\_track(2)
37. print('get long record:\n', cd1.get\_long\_record())
38. lstOfCDObjects.append(cd1)
40. print('\n\nTesting of class FileIO')
41. IO.FileIO.save\_inventory(file\_name, lstOfCDObjects)
42. print(IO.FileIO.load\_inventory(file\_name))
44. print('\n\nTesting ScreenIO class')
45. print('Main menu:')
46. IO.ScreenIO.print\_menu()
47. print('selection in menu: {}'.format(IO.ScreenIO.menu\_choice()))
48. print('Inventory:')
49. IO.ScreenIO.show\_inventory(lstOfCDObjects)
50. cd2 = DC.CD(2, 'test\_title\_2', 'cd\_artist\_2')
51. lstOfCDObjects.append(cd2)
52. print('Inventory:')
53. for item in lstOfCDObjects:
54. print(item)
55. cd\_idx = 1
56. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
57. print('\nSub Menu')
58. IO.ScreenIO.print\_CD\_menu()
59. print('selection in sub menu: {}'.format(IO.ScreenIO.menu\_CD\_choice()))
60. print('Tracks:')
61. IO.ScreenIO.show\_tracks(cd)
63. print('\n\nTesting Processing Classes')
64. PC.DataProcessor.add\_CD((3, 'Foreigner', 'Foreigner'), lstOfCDObjects)
65. print('Inventory:')
66. for item in lstOfCDObjects:
67. print(item)

1. Retrieved 2021-Dec-12 [↑](#footnote-ref-1)