John Hunter

March 23, 2020

Foundations of Programming: Python

Module 09

Knowledge Document

Introduction

The primary discussion topic for this module is Object Oriented Programming(OOP), and the extension of the fundamentals of OOP across a modularized file structure of execution. As an extension of the many of the ancillary topics the class has covered previously, the assignment09 tasks are the culmination of the course and employ most aspects of coding that were on full offer. Class definitions, processing and handling of data can now be distributed across a file system of the developer’s definition. The distribution requires much more coordination and preparation. To that end a brief mention of Unified Modeling Language(UML) is given along with resources for further investigation.

Materials Discussion

Modules have already been used in this class to help organize code. Depending on the Python version installed, there are many built-in modules available to the student. There are multiple methods of calling, or importing these modules for use in the script or application. The preferred method appears to be the assignment of aliases[1]. An alias can be assigned to shorten the typing required, or provide some clue as to the functionality being enabled by importing the module. For the modules that are not part of the standard Python install, separate files, that are not intended to be run independently will need to be created[2]. There are restrictions on where the modules may exist in order to be accessible to the main module[1]. In the assignment, all necessary modules, that are external to the Python installation, will be placed in the same file path as the main CD\_Inventory file. The Anaconda installation has features that will allow for the rules around file placement to be structured by the developer. Aliases can be standardized across the Python user community, e.g. pandas as pd, or numpy as np[1].

Main module is the only file that is usually intended to be directly executed. A simple way to handle this is to employ the system variable \_\_name\_\_ which checks to see if the module is main, and then can handle an error if the module is directly executed and is not \_\_main\_\_. Once the main module is executed, and the various modules are imported Python will Cache the imported modules with a .pyc file created automatically. If the files are deleted, the binary file is recreated when the main is executed again, a valid version is required to run when the external modules and their classes are called[2].

Some of the course materials are implementation specific, or are better explained alongside the coding challenge, including the test harness. Inheritance will be covered in the special assignment document.

Script Creation

The baseline has been provided as a starter in the module. There are “TODO” markers and some pseudocode through that represent the code and functionality that needed to be added[4]. Although other modules and assignments had similar starter packages, there is more, at the very least in terms of volume of code, provided here[2]. The structure of the assignment in this module is also somewhat different. The assignment instructions seem to indicate that we are handing in LAB 09-B[3]. This is unusual in that a solution exists, at least in part, for LAB 09-B[3]. The initial work then is to incorporate this code into the starter[2][3][4].

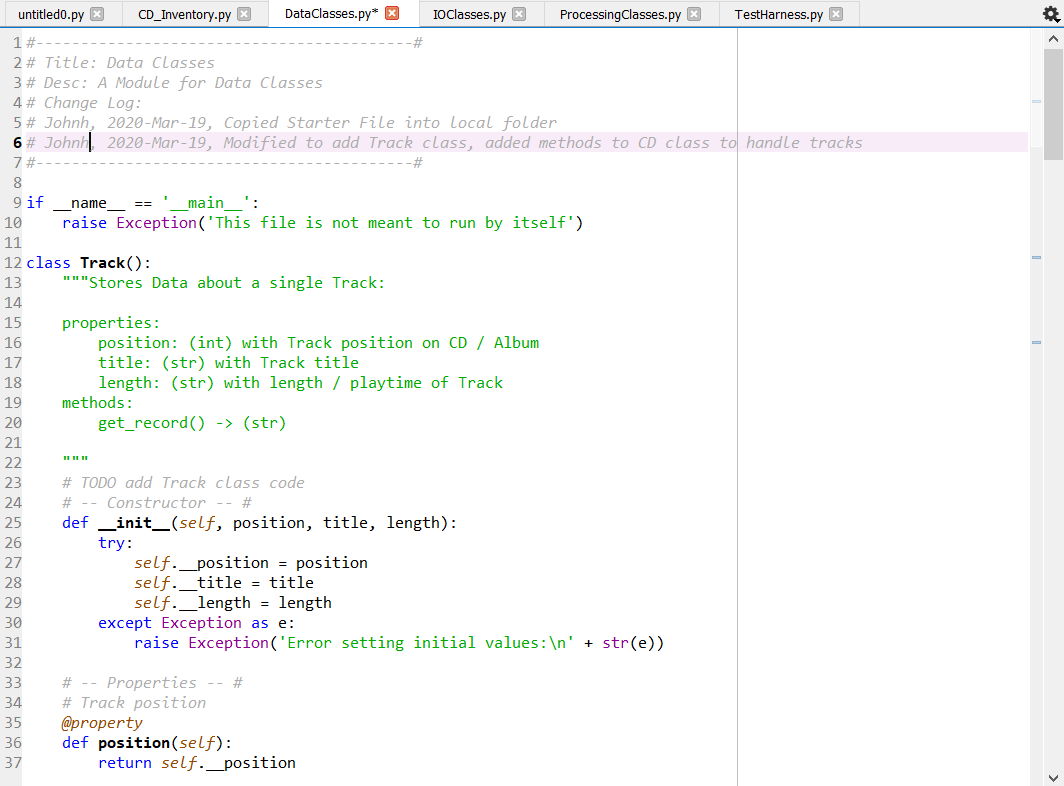


Figure Adding the provided code helps to start from a common baseline, where variable naming, and data structure can be established across all modules.

There are more straightforward aspects of writing the code. Replacing the header information, updating docstrings, and double checking the basic structure will not be shown. Some of the coding challenges here took quite a lot of effort, particularly, organizing the main module to work with the preferred structures of the predefined IO, data, and processing classes.

The basic copying over of the partial solution yielded code that would execute, however, only some of the functionality was present. This could have been due to copying errors or unhandled errors in the original code. This seemed immaterial, and the focus was on reorganizing the primary module. After some investigation of the best practices around the structure of main(), restructuring began.

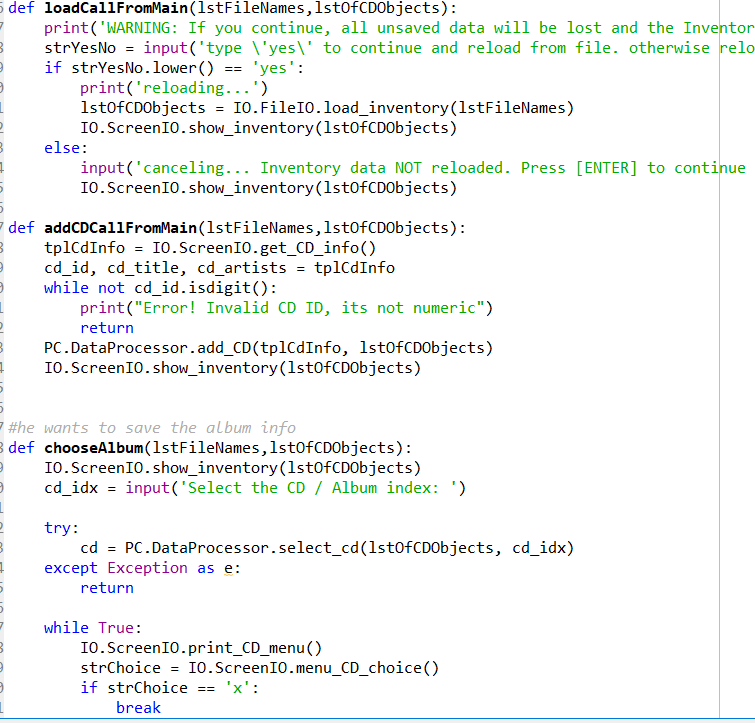
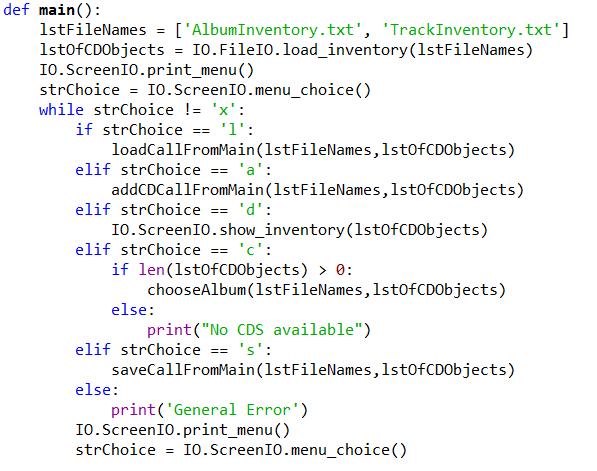


Figure 2 Added main() to organize the main module. Figure 3 Methods were added to main to organize the principle calls to the modules.

Main() is now able to function as a sort of summary of the entire script. Most of the functionality exists outside the main module, but none of it can be of any benefit to the user without the calls the main module makes. There is a loose relationship between the methods in the main() and the modules, addCDCallFromMain serves to add a CD above the lower level submenu, chooseAlbum serves to enable the use of that submenu, and the loadCallFromMain and SaveCallFromMain enable multiple parts of the saving and loading that are contained in the IOClasses module. Once this organization was added, and most of the functionality enabled, troubleshooting and error handling could begin.

There were many manual tests perform to correct remove track not working properly, error handled on rmv\_trk, error handling to enforce type int in track\_id. Additionally, there were a few extra bits here and then, for instance, the continues in the main while loop were not necessary, and so were removed. The sort\_tracks method was considered for removal, but it was left since the professor did not give clearance for its removal. Many of the original error handles ended the application, this has been prevented, the application now continues and asks for the data entry again. Crashes are not allowed, and the user is unable to crash the program in normal IO. No unit of measure is enforced for the length of the track, it can be entered in minutes, seconds, or jiffies if the user decides(there is likely an upper limit, so jiffies may not actually work). The track removal failed for a user selection of any track that did not have an entry, this has been corrected so that any track index without an entry is treated as no entry selected, at least the user facing component is.

Once all the errors that were identified were handled or rewritten, and the functionality was sufficiently completed, the test harness module was investigated. The code for the test harness module is ostensibly written to interrogate the modules that make up the CD\_Inventory application. The code of the test harness imports the 3 modules that are not tagged with the \_\_name\_\_==’\_\_main\_\_’ exception.

1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. **raise** Exception('This file is not meant to ran by itself')

Figure 4 The test harness and main modules are able to be run, other modules should not be run, but are called from either CD\_Inventory(Main), or the test harness.

The functionality of the test harness can be simplified as such, the classes contained in the three modules IO, Data, Processing, have their attributes populated, and have objects created from them. These data points are then printed onto the screen.

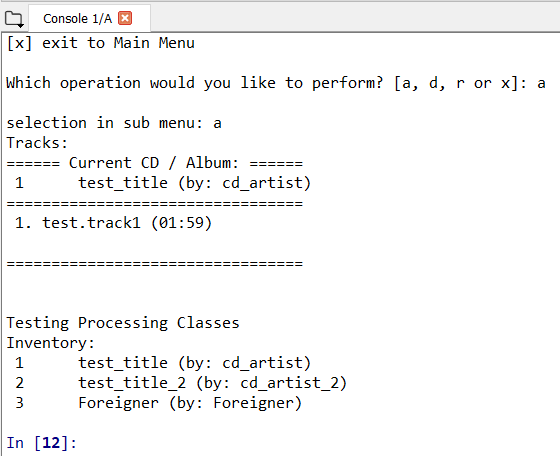
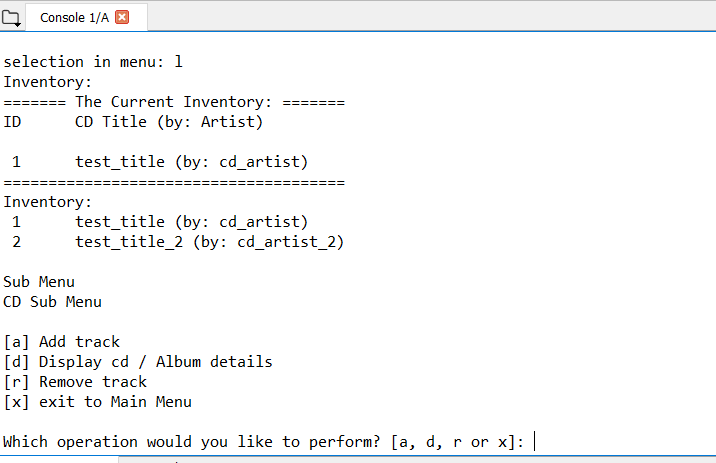


Figure 5 The load option is selected to begin testing. Figure 6 The tester selects to advance to the track Class testing at the lower level, then the test harness completes running. Other test paths must be selected to test all functionality.

The test harness is a useful construct to show how a test script can be authored that provides a measure of confidence that the script performs as expected. Here, no errors were found with the test harness, even though some were found with other testing procedures. The basic structure of the test harness is a decision tree. The user runs the program, there are six options, for most selected options there are 4 options at the second level. Since the ‘x’ selection does not take the user back to the main menu, there are a total of 24 paths, all of which were tested with no raised errors or crashes.

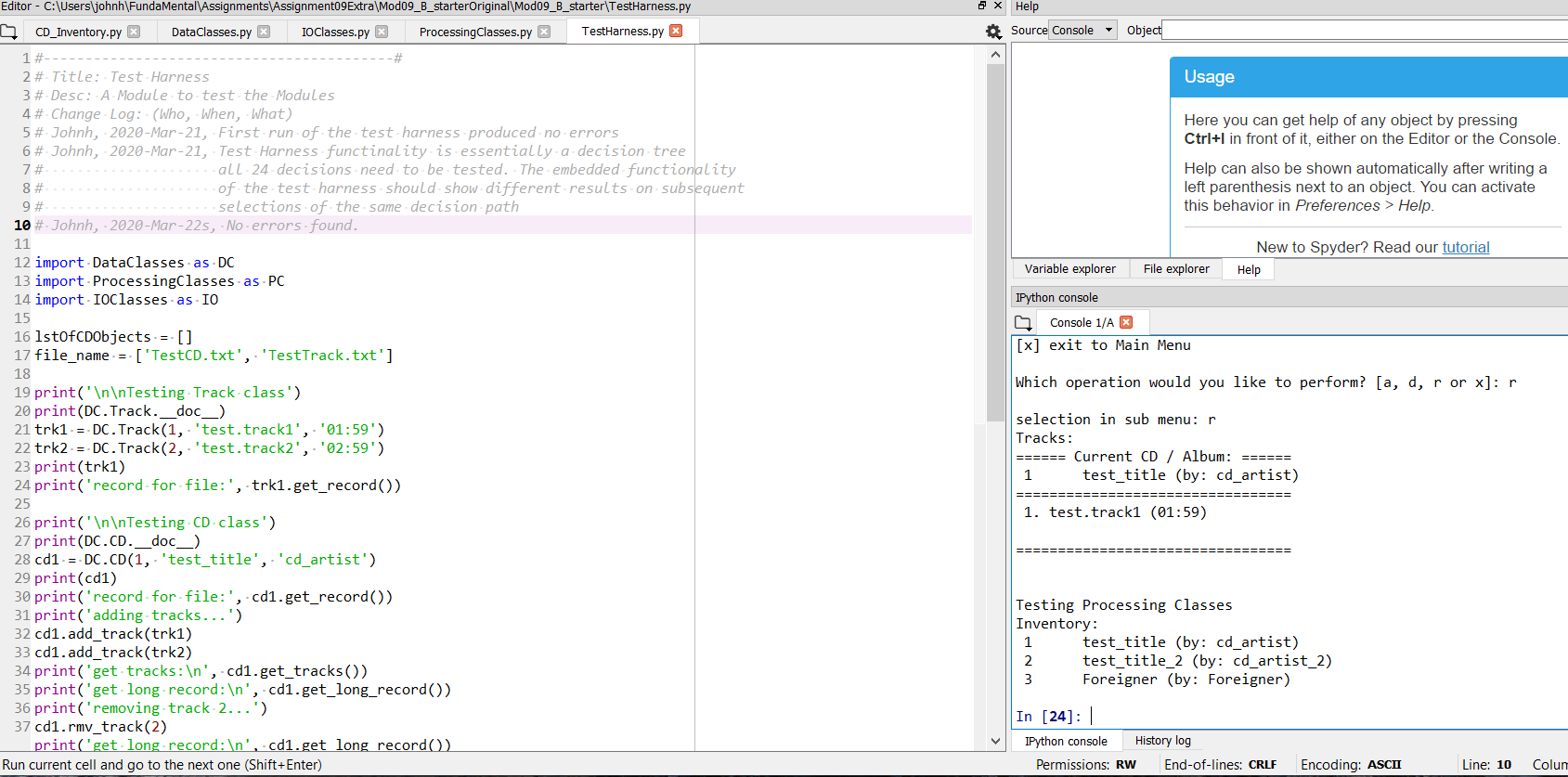


Figure 7 Test Harness did not catch some errors that were exposed through manual methods, e.g. passing bad data type.

Summary

There was an intense amount of refactoring to do once the code was reorganized. The refactoring would likely have been even more challenging without the modular approach. Modules allow for separation between functionality while handling many object types. This can be extended with inheritance, and likely will be covered in more depth in future classes. The partial solution provided in the labs was likely necessary given the scale of the code and the time frame under which this was to be completed. OOP is a wide-ranging field with many nuances, so important is the inheritance that modeling languages have been written to capture the inheritance and structure, like UML. The course covers the basics, but the core concepts will be covered in future classes

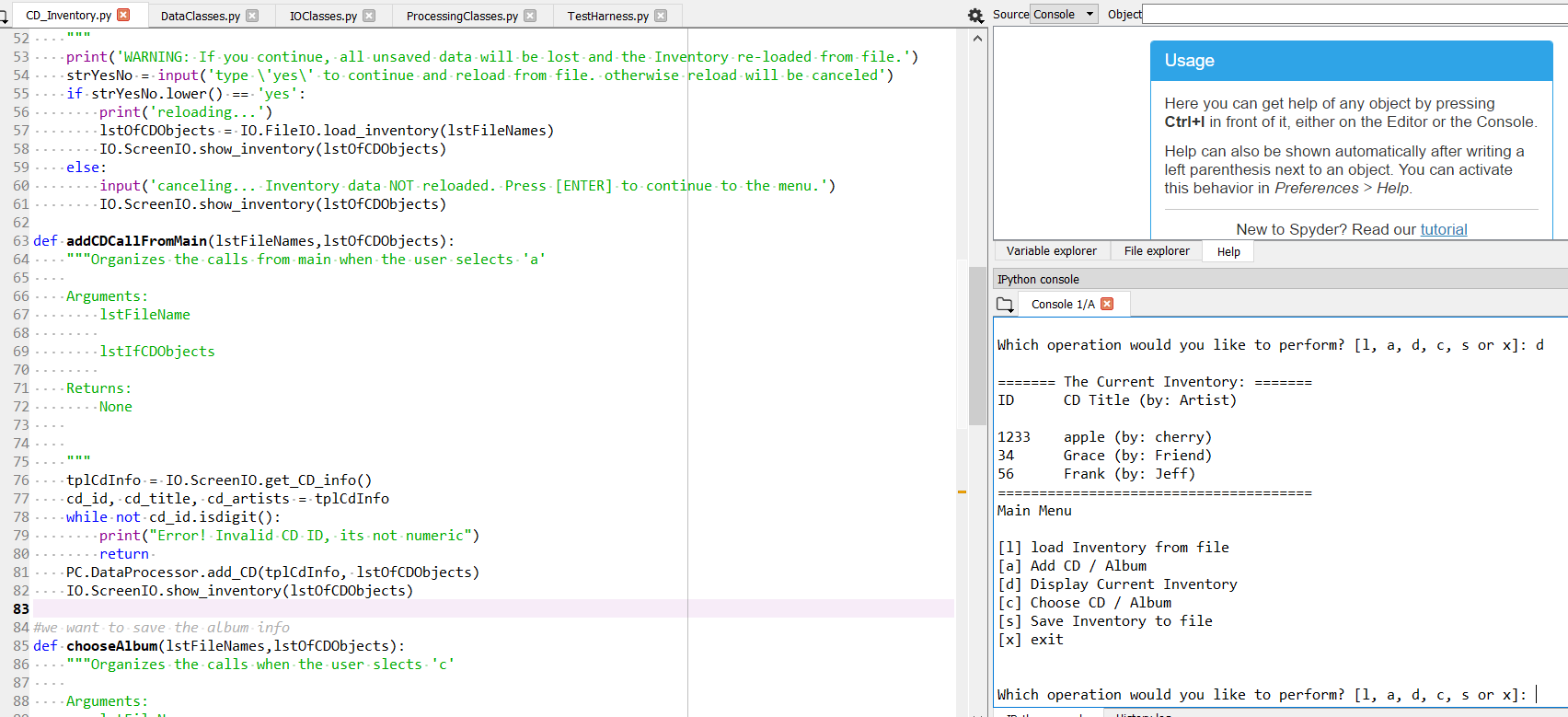


Figure 8 The code runs in Spyder and no errors were found.

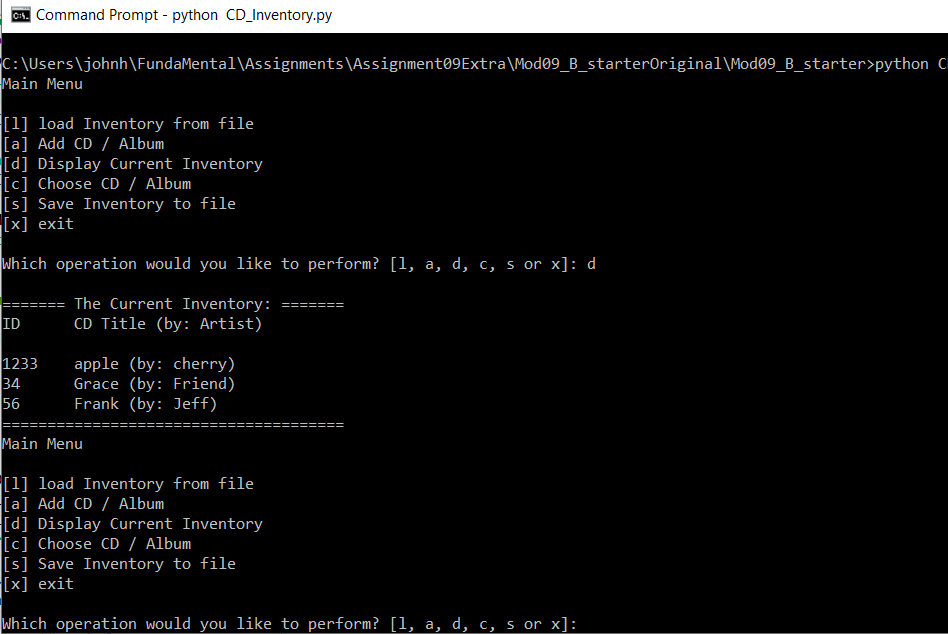


Figure 9 The code runs in the command prompt, and no errors are found.

Appendix

Planet B Syntax Highlighter

1. #------------------------------------------#
2. # Title: CD\_Inventory.py
3. # Desc: The CD Inventory App main Module
4. # Change Log:
5. # Johnh, 2020-Mar-19, Copied Starter Files into local folder
6. # Johnh, 2020-Mar-19, Added code from LAB09B solution
7. # Johnh, 2020-Mar-19, Added main()
8. # Johnh, 2020-Mar-20, Reorganized main into methods to handle user selections
9. # Johnh, 2020-Mar-20, Added error handles
10. # Johnh, 2020-Mar-22, Testing, no errors found
11. # Johnh, 2020-Mar-22, Updated docstrings
12. # Johnh, 2020-Mar-22, Test Harness Found no errors
13. #------------------------------------------#
15. **import** ProcessingClasses as PC
16. **import** IOClasses as IO
18. **def** main():
19. lstFileNames = ['AlbumInventory.txt', 'TrackInventory.txt']
20. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
21. IO.ScreenIO.print\_menu()
22. strChoice = IO.ScreenIO.menu\_choice()
23. **while** strChoice != 'x':
24. **if** strChoice == 'l':
25. loadCallFromMain(lstFileNames,lstOfCDObjects)
26. **elif** strChoice == 'a':
27. addCDCallFromMain(lstFileNames,lstOfCDObjects)
28. **elif** strChoice == 'd':
29. IO.ScreenIO.show\_inventory(lstOfCDObjects)
30. **elif** strChoice == 'c':
31. **if** len(lstOfCDObjects) > 0:
32. chooseAlbum(lstFileNames,lstOfCDObjects)
33. **else**:
34. **print**("No CDS available")
35. **elif** strChoice == 's':
36. saveCallFromMain(lstFileNames,lstOfCDObjects)
37. **else**:
38. **print**('General Error')
39. IO.ScreenIO.print\_menu()
40. strChoice = IO.ScreenIO.menu\_choice()
42. **def** loadCallFromMain(lstFileNames,lstOfCDObjects):
43. """Organizes the 'l' user selection
45. Arguments:
46. lstFileNames
48. lstOfCDObjects
50. Returns:
51. None
52. """
53. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
54. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
55. **if** strYesNo.lower() == 'yes':
56. **print**('reloading...')
57. lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
58. IO.ScreenIO.show\_inventory(lstOfCDObjects)
59. **else**:
60. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
61. IO.ScreenIO.show\_inventory(lstOfCDObjects)
63. **def** addCDCallFromMain(lstFileNames,lstOfCDObjects):
64. """Organizes the calls from main when the user selects 'a'
66. Arguments:
67. lstFileName
69. lstIfCDObjects
71. Returns:
72. None

75. """
76. tplCdInfo = IO.ScreenIO.get\_CD\_info()
77. cd\_id, cd\_title, cd\_artists = tplCdInfo
78. **while** **not** cd\_id.isdigit():
79. **print**("Error! Invalid CD ID, its not numeric")
80. **return**
81. PC.DataProcessor.add\_CD(tplCdInfo, lstOfCDObjects)
82. IO.ScreenIO.show\_inventory(lstOfCDObjects)
84. #we want to save the album info
85. **def** chooseAlbum(lstFileNames,lstOfCDObjects):
86. """Organizes the calls when the user slects 'c'
88. Arguments:
89. lstFileName
91. lstIfCDObjects
93. Returns:
94. None
96. """
97. IO.ScreenIO.show\_inventory(lstOfCDObjects)
98. cd\_idx = input('Select the CD / Album index: ')
100. **try**:
101. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
102. **except** Exception as e:
103. **return**
105. **while** True:
106. IO.ScreenIO.print\_CD\_menu()
107. strChoice = IO.ScreenIO.menu\_CD\_choice()
108. **if** strChoice == 'x':
109. **break**
110. **if** strChoice == 'a':    #modify track
111. tplTrkInfo = IO.ScreenIO.get\_track\_info()
112. trkID, trkTitle, trkLength = tplTrkInfo
113. **if** trkID.isdigit() **and** 0 <= int(trkID) - 1:
114. PC.DataProcessor.add\_track(tplTrkInfo, cd)
115. **else**:
116. **print**("Invalid track ID")
117. **elif** strChoice == 'd':
118. **try**:
119. cd.get\_tracks()
120. IO.ScreenIO.show\_tracks(cd)
121. **except** Exception as e:
122. **print**(e)
123. **elif** strChoice == 'r':
124. **try**:
125. cd.get\_tracks()
126. IO.ScreenIO.show\_tracks(cd)
127. trk\_idx = int(input('Select the Track index: '))
128. cd.rmv\_track(trk\_idx)
129. **except** Exception as e:
130. **print**(e)
131. **else**:
132. **print**('General Error')
134. **def** saveCallFromMain(lstFileNames,lstOfCDObjects):
135. """Organizes the calls when the user selects 's'
137. Arguments:
138. lstFileName
140. lstIfCDObjects
142. Returns:
143. None
145. """
146. IO.ScreenIO.show\_inventory(lstOfCDObjects)
147. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
148. **if** strYesNo == 'y':
149. IO.FileIO.save\_inventory(lstFileNames, lstOfCDObjects)
150. **else**:
151. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')

154. main()
155. #------------------------------------------#
156. # Title: Data Classes
157. # Desc: A Module for Data Classes
158. # Change Log:
159. # Johnh, 2020-Mar-19, Copied Starter Files into local folder
160. # Johnh, 2020-Mar-19, Added code from LAB09B solution
161. # Johnh, 2020-Mar-19, Added error handles
162. # Johnh, 2020-Mar-21, Testing, no errors found
163. # Johnh, 2020-Mar-21, Updated docstrings
164. # Johnh, 2020-Mar-22, Test Harness Found no errors
165. #------------------------------------------#
167. **if** \_\_name\_\_ == '\_\_main\_\_':
168. **raise** Exception('This file is not meant to run by itself')
170. **class** Track():
171. """Stores Data about a single Track:
173. properties:
174. position: (int) with Track position on CD / Album
175. title: (str) with Track title
176. length: (str) with length / playtime of Track
177. methods:
178. get\_record() -> (str)
179. """
180. # -- Constructor -- #
181. **def** \_\_init\_\_(self, position, title, length):
182. **try**:
183. self.\_\_position = position
184. self.\_\_title = title
185. self.\_\_length = length
186. **except** Exception as e:
187. **raise** Exception('Error setting initial values:\n' + str(e))
189. # -- Properties -- #
190. # Track position
191. @property
192. **def** position(self):
193. **return** self.\_\_position
195. @position.setter
196. **def** position(self, value):
197. **if** type(value) == int:
198. **if** value < 1:
199. **raise** Exception('Position can\'t be less than 1!')
200. self.\_\_position = value
201. **else**:
202. **raise** Exception('Position needs to be integer')
203. # Track title
204. @property
205. **def** title(self):
206. **return** self.\_\_title
208. @title.setter
209. **def** title(self, value):
210. **if** type(value) == str:
211. self.\_\_title = value
212. **else**:
213. **raise** Exception('Title needs to be string')
215. # Track length
216. @property
217. **def** length(self):
218. **return** self.\_\_length
220. @length.setter
221. **def** length(self, value):
222. **if** type(value) == str:
223. self.\_\_length = value
224. **else**:
225. **raise** Exception('Length needs to be string')
227. # -- Methods -- #
228. **def** \_\_str\_\_(self):
229. """Returns Track details as formatted string"""
230. **return** '{:>2}. {} ({})'.format(self.position, self.title, self.length)

233. **def** get\_record(self) -> str:
234. """Returns: Track record formatted for saving to file"""
235. **return** '{},{},{}\n'.format(self.position, self.title, self.length)

238. **class** CD:
239. """Stores data about a CD / Album:
241. properties:
242. cd\_id: (int) with CD  / Album ID
243. cd\_title: (string) with the title of the CD / Album
244. cd\_artist: (string) with the artist of the CD / Album
245. cd\_tracks: (list) with track objects of the CD / Album
246. methods:
247. get\_record() -> (str)
248. add\_track(track) -> None
249. rmv\_track(int) -> None
250. get\_tracks() -> (str)
251. get\_long\_record() -> (str)
252. """
253. # -- Constructor -- #
254. **def** \_\_init\_\_(self, cd\_id: int, cd\_title: str, cd\_artist: str) -> None:
255. """Set ID, Title and Artist of a new CD Object"""
256. #    -- Attributes  -- #
257. **try**:
258. self.\_\_cd\_id = int(cd\_id)
259. self.\_\_cd\_title = str(cd\_title)
260. self.\_\_cd\_artist = str(cd\_artist)
261. self.\_\_tracks = []
262. **except** Exception as e:
263. **raise** Exception('Error setting initial values:\n' + str(e))
265. # -- Properties -- #
266. # CD ID
267. @property
268. **def** cd\_id(self):
269. **return** self.\_\_cd\_id
271. @cd\_id.setter
272. **def** cd\_id(self, value):
273. **try**:
274. self.\_\_cd\_id = int(value)
275. **except** Exception:
276. **raise** Exception('ID needs to be Integer')
278. # CD title
279. @property
280. **def** cd\_title(self):
281. **return** self.\_\_cd\_title
283. @cd\_title.setter
284. **def** cd\_title(self, value):
285. **try**:
286. self.\_\_cd\_title = str(value)
287. **except** Exception:
288. **raise** Exception('Title needs to be String!')
290. # CD artist
291. @property
292. **def** cd\_artist(self):
293. **return** self.\_\_cd\_artist
295. @cd\_artist.setter
296. **def** cd\_artist(self, value):
297. **try**:
298. self.\_\_cd\_artist = str(value)
299. **except** Exception:
300. **raise** Exception('Artist needs to be String!')
302. # CD tracks
303. @property
304. **def** cd\_tracks(self):
305. **return** self.\_\_tracks
307. # -- Methods -- #
308. **def** \_\_str\_\_(self):
309. """Returns: CD details as formatted string"""
310. **return** '{:>2}\t{} (by: {})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
312. **def** get\_record(self):
313. """Returns: CD record formatted for saving to file"""
314. **return** '{},{},{}\n'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
316. **def** add\_track(self, track: Track) -> None:
317. """Adds a track to the CD / Album
319. Args:
320. track (Track): Track object to be added to CD / Album.
322. Returns:
323. None.
324. """
325. self.\_\_tracks.append(track)
326. self.\_\_sort\_tracks()
328. **def** rmv\_track(self, track\_id: int) -> None:
329. """Removes the track identified by track\_id from Album
331. Args:
332. track\_id (int): ID of track to be removed.
334. Returns:
335. None.
336. """
337. **if** **not** isinstance(track\_id, int):
338. **raise** ValueError(track\_id)
340. **del** self.\_\_tracks[track\_id - 1]
341. self.\_\_sort\_tracks()
343. **def** \_\_sort\_tracks(self):
344. """Sorts the tracks using Track.position. Fills blanks with None"""
345. n = len(self.\_\_tracks)
346. **for** track **in** self.\_\_tracks:
347. **if** (track **is** **not** None) **and** (n < track.position):
348. n = track.position
349. tmp\_tracks = [None] \* n
350. **for** track **in** self.\_\_tracks:
351. **if** track **is** **not** None:
352. tmp\_tracks[track.position - 1] = track
353. self.\_\_tracks = tmp\_tracks
355. **def** get\_tracks(self) -> str:
356. """Returns a string list of the tracks saved for the Album
358. Raises:
359. Exception: If no tracks are saved with album.
361. Returns:
362. result (string):formatted string of tracks.
363. """
364. self.\_\_sort\_tracks()
365. **if** len(self.\_\_tracks) < 1:
366. **raise** Exception('No tracks saved for this Album')
367. result = ''
368. **for** track **in** self.\_\_tracks:
369. **if** track **is** None:
370. result += 'No Information for this track\n'
371. **else**:
372. result += str(track) + '\n'
373. **return** result
375. **def** get\_long\_record(self) -> str:
376. """gets a formatted long record of the Album: Album information plus track details
378. Returns:
379. result (string): Formatted information about ablum and its tracks.
380. """
381. result = self.get\_record() + '\n'
382. result += self.get\_tracks() + '\n'
383. **return** result
384. #------------------------------------------#
385. # Title: IO Classes
386. # Desc: A Module for IO Classes
387. # Change Log:
388. # Johnh, 2020-Mar-19, Copied Starter Files into local folder
389. # Johnh, 2020-Mar-19, Added code from LAB09B solution
390. # Johnh, 2020-Mar-19, Added error handles
391. # Johnh, 2020-Mar-21, Testing, no errors found
392. # Johnh, 2020-Mar-21, Updated docstrings
393. # Johnh, 2020-Mar-22, Test Harness Found no errors
394. #------------------------------------------#
396. **if** \_\_name\_\_ == '\_\_main\_\_':
397. **raise** Exception('This file is not meant to run by itself')
399. **import** DataClasses as DC
400. **import** ProcessingClasses as PC
402. **class** FileIO:
403. """Processes data to and from file:
405. properties:
407. methods:
408. save\_inventory(file\_name, lst\_Inventory): -> None
409. load\_inventory(file\_name): -> (a list of CD objects)
410. """
412. @staticmethod
413. **def** save\_inventory(file\_name: list, lst\_Inventory: list) -> None:
414. """
416. Args:
417. file\_name (list): list of file names [CD Inventory, Track Inventory] that hold the data.
418. lst\_Inventory (list): list of CD objects.
420. Returns:
421. None.
422. """
424. file\_name\_CD = file\_name[0]
425. file\_name\_trk = file\_name[1]
426. **try**:
427. with open(file\_name\_CD, 'w') as file:
428. **for** disc **in** lst\_Inventory:
429. file.write(disc.get\_record())
430. with open(file\_name\_trk, 'w') as file:
431. **for** disc **in** lst\_Inventory:
432. tracks = disc.cd\_tracks
433. disc\_id = disc.cd\_id
434. **for** trk **in** tracks:
435. **if** trk **is** **not** None:
436. record = '{},{}'.format(disc\_id, trk.get\_record())
437. file.write(record)
438. **except** Exception as e:
439. **print**('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')
441. @staticmethod
442. **def** load\_inventory(file\_name: list) -> list:
443. """Reads in file\_name list and creates the CD objects and Track Objects
444. and returns the 2 lists of objects.
446. Args:
447. file\_name (list): list of file names [CD Inventory, Track Inventory]
448. that hold the data.
450. Returns:
451. lst\_Inventory: list of CD objects.
452. """
454. lst\_Inventory = []
455. file\_name\_CD = file\_name[0]
456. file\_name\_trk = file\_name[1]
457. **try**:
458. with open(file\_name\_CD, 'r') as file:
459. **for** line **in** file:
460. data = line.strip().split(',')
461. row = DC.CD(data[0], data[1], data[2])
462. lst\_Inventory.append(row)
463. with open(file\_name\_trk, 'r') as file:
464. **for** line **in** file:
465. data = line.strip().split(',')
466. cd = PC.DataProcessor.select\_cd(lst\_Inventory, int(data[0]))
467. track = DC.Track(int(data[1]), data[2], data[3])
468. cd.add\_track(track)
469. **except** Exception as e:
470. **print**('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')
471. **return** lst\_Inventory
473. **class** ScreenIO:
474. """Handling Input / Output
476. properties:
478. methods:
479. print\_menu():
480. menu\_chioce():
481. print\_CD\_menu():
482. menu\_CD\_choice():
483. show\_inventory(table):
484. show\_tracks(cd):
485. get\_CD\_info():
486. get\_track\_info():
487. """
489. @staticmethod
490. **def** print\_menu():
491. """Displays a menu of choices to the user
493. Args:
494. None.
496. Returns:
497. None.
498. """
500. **print**('Main Menu\n\n[l] load Inventory from file\n[a] Add CD / Album\n[d] Display Current Inventory')
501. **print**('[c] Choose CD / Album\n[s] Save Inventory to file\n[x] exit\n')
503. @staticmethod
504. **def** menu\_choice():
505. """Gets user input for menu selection
507. Args:
508. None.
510. Returns:
511. choice (string): a lower case sting of the users input out of the choices l, a, d, c, s or x
512. """
514. choice = ' '
515. **while** choice **not** **in** ['l', 'a', 'd', 'c', 's', 'x']:
516. choice = input('Which operation would you like to perform? [l, a, d, c, s or x]: ').lower().strip()
517. **print**()  # Add extra space for layout
518. **return** choice
520. @staticmethod
521. **def** print\_CD\_menu():
522. """Displays a sub menu of choices for CD / Album to the user
524. Args:
525. None.
527. Returns:
528. None.
529. """
531. **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')
533. @staticmethod
534. **def** menu\_CD\_choice():
535. """Gets user input for CD sub menu selection
537. Args:
538. None.
540. Returns:
541. choice (string): a lower case sting of the users input out of the choices a, d, r or x
542. """
544. choice = ' '
545. **while** choice **not** **in** ['a', 'd', 'r', 'x']:
546. choice = input('Which operation would you like to perform? [a, d, r or x]: ').lower().strip()
547. **print**()  # Add extra space for layout
548. **return** choice
550. @staticmethod
551. **def** show\_inventory(table):
552. """Displays current inventory table
554. Args:
555. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
557. Returns:
558. None.
559. """
561. **print**('======= The Current Inventory: =======')
562. **print**('ID\tCD Title (by: Artist)\n')
563. **for** row **in** table:
564. **print**(row)
565. **print**('======================================')
567. @staticmethod
568. **def** show\_tracks(cd):
569. """Displays the Tracks on a CD / Album
571. Args:
572. cd (CD): CD object.
574. Returns:
575. None.
576. """
577. **print**('====== Current CD / Album: ======')
578. **print**(cd)
579. **print**('=================================')
580. **print**(cd.get\_tracks())
581. **print**('=================================')
583. @staticmethod
584. **def** get\_CD\_info():
585. """function to request CD information from User to add CD to inventory
587. Returns:
588. cdId (string): Holds the ID of the CD dataset.
589. cdTitle (string): Holds the title of the CD.
590. cdArtist (string): Holds the artist of the CD.
591. """
592. cdId = input('Enter ID: ').strip()
593. cdTitle = input('What is the CD\'s title? ').strip()
594. cdArtist = input('What is the Artist\'s name? ').strip()
595. **return** cdId, cdTitle, cdArtist
597. @staticmethod
598. **def** get\_track\_info():
599. """function to request Track information from User to add Track to CD / Album
601. Returns:
602. trkId (string): Holds the ID of the Track dataset.
603. trkTitle (string): Holds the title of the Track.
604. trkLength (string): Holds the length (time) of the Track.
605. """
607. trkId = input('Enter Position on CD / Album: ').strip()
608. trkTitle = input('What is the Track\'s title? ').strip()
609. trkLength = input('What is the Track\'s length? ').strip()
610. **return** trkId, trkTitle, trkLength
611. #------------------------------------------#
612. # Title: Processing Classes
613. # Desc: A Module for processing Classes
614. # Change Log:
615. # Johnh, 2020-Mar-19, Copied Starter Files into local folder
616. # Johnh, 2020-Mar-19, Added code from LAB09B solution
617. # Johnh, 2020-Mar-19, Added error handles
618. # Johnh, 2020-Mar-21, Testing, no errors found
619. # Johnh, 2020-Mar-21, Updated docstrings
620. # Johnh, 2020-Mar-22, Test Harness Found no errors
621. #------------------------------------------#
623. **if** \_\_name\_\_ == '\_\_main\_\_':
624. **raise** Exception('This file is not meant to ran by itself')
626. **import** DataClasses as DC
628. **class** DataProcessor:
629. """Processing the data in the application, allows user to add a CD to inventory
630. select a target CD to get detailed information, and add a track at the detailed
631. information level.
633. properties:
635. methods:
636. add\_CD(CDInfo, table):
637. select\_cd(table: list, cd\_idx: int) -> DC.CD:
638. add\_track(track\_info: tuple, cd: DC.CD) -> None:
639. """
641. @staticmethod
642. **def** add\_CD(CDInfo, table):
643. """function to add CD info in CDinfo to the inventory table.
645. Args:
646. CDInfo (tuple): Holds information (ID, CD Title, CD Artist) to be added to inventory.
647. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
649. Returns:
650. None.
651. """
653. cdId, title, artist = CDInfo
654. **try**:
655. cdId = int(cdId)
656. **except**:
657. **raise** Exception('ID must be an Integer!')
658. row = DC.CD(cdId, title, artist)
659. table.append(row)
661. @staticmethod
662. **def** select\_cd(table: list, cd\_idx: int) -> DC.CD:
663. """selects a CD object out of table that has the ID cd\_idx
665. Args:
666. table (list): Inventory list of CD objects.
667. cd\_idx (int): id of CD object to return
669. Raises:
670. Exception: If id is not in list.
672. Returns:
673. row (DC.CD): CD object that matches cd\_idx
674. """
676. **try**:
677. cd\_idx = int(cd\_idx)
678. **except** ValueError as e:
679. **print**('ID is not an Integer!')
680. **print**(e.\_\_doc\_\_)
681. **for** row **in** table:
682. **if** row.cd\_id == cd\_idx:
683. **return** row
684. **raise** Exception('This CD / Album index does not exist')
686. @staticmethod
687. **def** add\_track(track\_info: tuple, cd: DC.CD) -> None:
688. """adds a Track object with attributes in track\_info to cd
690. Args:
691. track\_info (tuple): Tuple containing track info (position, title, Length).
692. cd (DC.CD): cd object the tarck gets added to.
694. Raises:
695. Exception: DESCraised in case position is not an integer.
697. Returns:
698. None: DESCRIPTION.
699. """
701. trkPos, trkTitle, trkLength = track\_info
702. **try**:
703. trkPos = int(trkPos)
704. **except**:
705. **raise** Exception('Position must be an Integer')
706. track = DC.Track(trkPos, trkTitle, trkLength)
707. cd.add\_track(track)
708. #------------------------------------------#
709. # Title: Test Harness
710. # Desc: A Module to test the Modules
711. # Change Log: (Who, When, What)
712. # Johnh, 2020-Mar-21, First run of the test harness produced no errors
713. # Johnh, 2020-Mar-21, Test Harness functinality is essentially a decision tree
714. #                     all 24 decisions need to be tested. The embedded functionality
715. #                     of the test harness should show different results on subsequent
716. #                     selections of the same decision path
717. # Johnh, 2020-Mar-22s, No errors found.
719. **import** DataClasses as DC
720. **import** ProcessingClasses as PC
721. **import** IOClasses as IO
723. lstOfCDObjects = []
724. file\_name = ['TestCD.txt', 'TestTrack.txt']
726. **print**('\n\nTesting Track class')
727. **print**(DC.Track.\_\_doc\_\_)
728. trk1 = DC.Track(1, 'test.track1', '01:59')
729. trk2 = DC.Track(2, 'test.track2', '02:59')
730. **print**(trk1)
731. **print**('record for file:', trk1.get\_record())
733. **print**('\n\nTesting CD class')
734. **print**(DC.CD.\_\_doc\_\_)
735. cd1 = DC.CD(1, 'test\_title', 'cd\_artist')
736. **print**(cd1)
737. **print**('record for file:', cd1.get\_record())
738. **print**('adding tracks...')
739. cd1.add\_track(trk1)
740. cd1.add\_track(trk2)
741. **print**('get tracks:\n', cd1.get\_tracks())
742. **print**('get long record:\n', cd1.get\_long\_record())
743. **print**('removing track 2...')
744. cd1.rmv\_track(2)
745. **print**('get long record:\n', cd1.get\_long\_record())
746. lstOfCDObjects.append(cd1)
748. **print**('\n\nTesting of class FileIO')
749. IO.FileIO.save\_inventory(file\_name, lstOfCDObjects)
750. **print**(IO.FileIO.load\_inventory(file\_name))
752. **print**('\n\nTesting ScreenIO class')
753. **print**('Main menu:')
754. IO.ScreenIO.print\_menu()
755. **print**('selection in menu: {}'.format(IO.ScreenIO.menu\_choice()))
756. **print**('Inventory:')
757. IO.ScreenIO.show\_inventory(lstOfCDObjects)
758. cd2 = DC.CD(2, 'test\_title\_2', 'cd\_artist\_2')
759. lstOfCDObjects.append(cd2)
760. **print**('Inventory:')
761. **for** item **in** lstOfCDObjects:
762. **print**(item)
763. cd\_idx = 1
764. cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
765. **print**('\nSub Menu')
766. IO.ScreenIO.print\_CD\_menu()
767. **print**('selection in sub menu: {}'.format(IO.ScreenIO.menu\_CD\_choice()))
768. **print**('Tracks:')
769. IO.ScreenIO.show\_tracks(cd)
771. **print**('\n\nTesting Processing Classes')
772. PC.DataProcessor.add\_CD((3, 'Foreigner', 'Foreigner'), lstOfCDObjects)
773. **print**('Inventory:')
774. **for** item **in** lstOfCDObjects:
775. **print**(item)

References

1. https://youtu.be/UgPh-0sEjTA
2. FDN\_Py\_Module\_09.pdf
3. Assignment09
4. Mod09\_B\_starter
5. https://realpython.com/python-main-function/