Shaked Bason

September-09-2020

Foundations of Programming (Python)

Assignment 09

Assignment 09

Instruction

In this assignment, questions such as “What is the difference between a class and module?,” and “How do I connect one module to another?” were answered. Throughout the assignment, I learned what is the "\_\_name\_\_ " System Variable and what is class inheritance.

Github Link for my Assignment - <https://github.com/shakba/Assignment09>

Module 9 and Labs – Step by step

The Main Module

python programs often use two or more files, however typically only one file is meant to be run directly. Any script that is run directly at the start of a program is called the “Main Module”.

The \_\_name\_\_ System Variable

Is a string type attribute that contains the name of the module. We usually get simple strings. In the main script, it returns ‘\_\_main\_\_’. Inside a module, it returns the module name.

Organizing Modules

DataClass.py - supports the main module. This DataClasses again is going to our CDInventory.py, includes a class for a CD, properties and methods like \_\_str\_\_() and get\_record().

IOClasses.py - This script also supports the main module. This module includes FileIO and ScrenIO classes.

ProcessingClasses.py - This script is also a supporter of the main module. This script includes a DataProcessor class which contains different methods of processing data like add\_CD.

TestHarness.py-module which includes all of the tests. We want to test the other modules so this is why we import the other modules(DataClass, IOClasses and ProcessingClasses) creating an empty list object and running through multiple tests. Test and use specific dedicated test cases for each method

Creating a program

I noted what the code does and added it to this document using the [planet](http://planetb.ca/syntax-highlight-word)-b website.

Finally, I created my code for this assignment (Appendix01), our known and familiar CDInventory. I modify my code to work with some new instructions.

**Appendix #1**

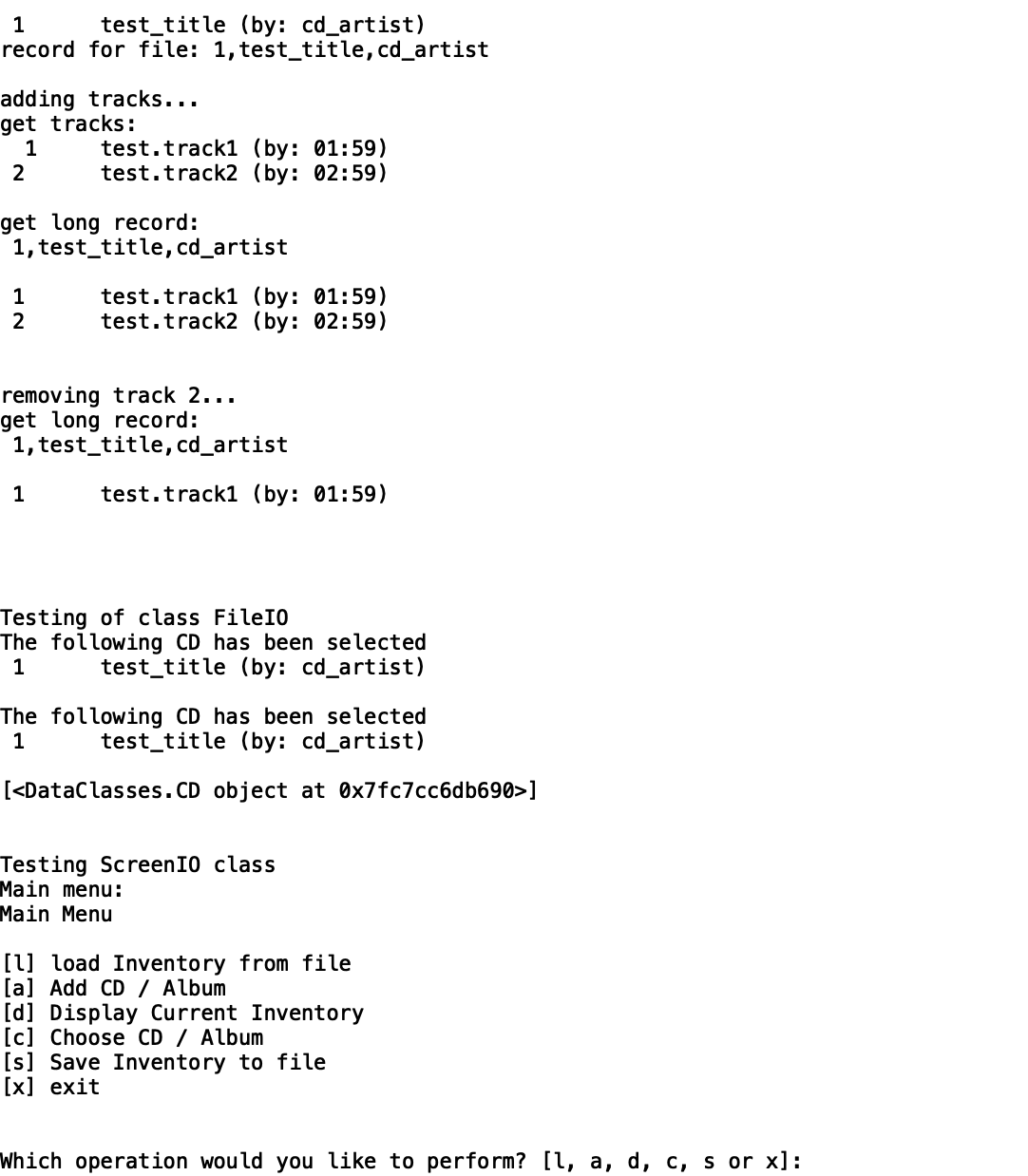
****

Figure 1- TestHarness.py output

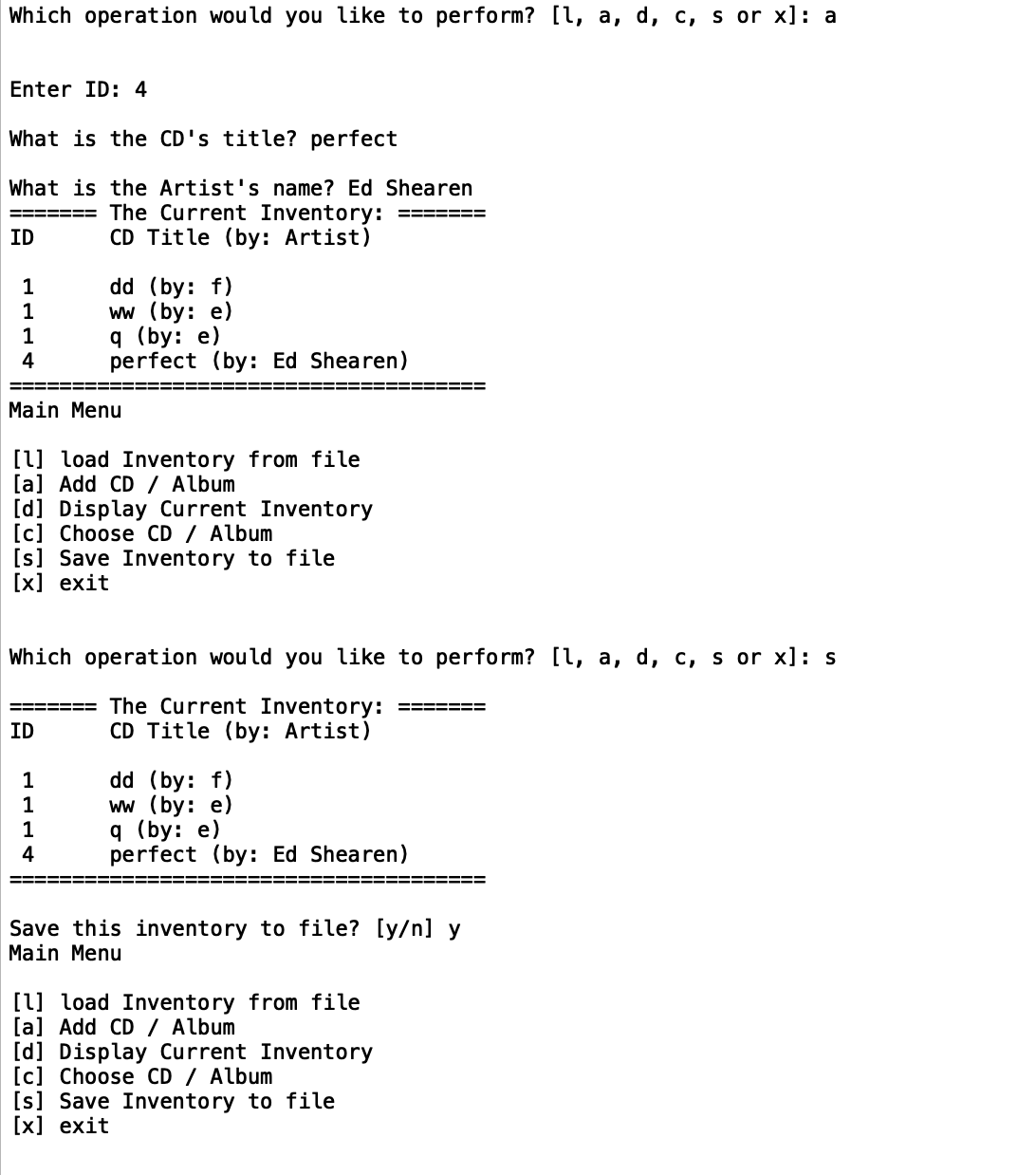


Figure 2- CD\_Inventory output

Summary

Module 9 taught me more about working with OOP, allowed objects to communicate with each other. I continued improving my usage of complex objects.

I used Spyder as my IDE on this assignment.

Appendix

#1 CDInventory.py

* #------------------------------------------#
* # Title: CD\_Inventory.py
* # Desc: The CD Inventory App main Module
* # Change Log: (Who, When, What)
* # DBiesinger, 2030-Jan-01, Created File
* # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
* #------------------------------------------#
* **import** ProcessingClasses as PC
* **import** IOClasses as IO
* lstFileNames = ['AlbumInventory.txt', 'TrackInventory.txt']
* lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
* **while** True:
* IO.ScreenIO.print\_menu()
* strChoice = IO.ScreenIO.menu\_choice()
* **if** strChoice == 'x':
* **break**
* **if** strChoice == 'l':
* **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
* strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
* **if** strYesNo.lower() == 'yes':
* **print**('reloading...')
* lstOfCDObjects = IO.FileIO.load\_inventory(lstFileNames)
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* **else**:
* input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* **continue** # start loop back at top.
* **elif** strChoice == 'a':
* tplCdInfo = IO.ScreenIO.get\_CD\_info()
* PC.DataProcessor.add\_CD(tplCdInfo, lstOfCDObjects)
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* **continue** # start loop back at top.
* **elif** strChoice == 'd':
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* **continue** # start loop back at top.
* **elif** strChoice == 'c':
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* cd\_idx = input('Select the CD / Album index: ')
* cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
* **print**('\nselected cd:', cd, '\n')
* **while** True:
* IO.ScreenIO.print\_CD\_menu()
* str\_Choice = IO.ScreenIO.menu\_CD\_choice()
* **if** str\_Choice == 'x':
* **break**
* **if** str\_Choice == 'a':
* track\_info = IO.ScreenIO.get\_track\_info()
* PC.DataProcessor.add\_track(track\_info, cd)
* **elif** str\_Choice == 'd':
* IO.ScreenIO.show\_tracks(cd)
* **elif** str\_Choice == 'r':
* IO.ScreenIO.show\_tracks(cd)
* track\_idx = int(input ('Select the track you wish to deleted:'))
* cd.rmv\_track(track\_idx)
* **else**:
* **print**('Error')
* **elif** strChoice == 's':
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
* **if** strYesNo == 'y':
* IO.FileIO.save\_inventory(lstFileNames, lstOfCDObjects)
* **else**:
* input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
* **continue** # start loop back at top.
* **else**:
* **print**('General Error')

DataClasses

* #------------------------------------------#
* # Title: Data Classes
* # Desc: A Module for Data Classes
* # Change Log: (Who, When, What)
* # DBiesinger, 2030-Jan-01, Created File
* # DBiesinger, 2030-Jan-02, Modified to add Track class, added methods to CD class to handle tracks
* # shakedbason, 2020-Sep-7, Modified
* #------------------------------------------#
* **if** \_\_name\_\_ == '\_\_main\_\_':
* **raise** Exception('This file is not meant to run by itself')
* **class** Track():
* """Stores Data about a single Track:
* properties:
* position: (int) with Track position on CD / Album
* title: (str) with Track title
* length: (str) with length / playtime of Track
* methods:
* get\_record() -> (str)
* """
* # -- Constructor -- #
* **def** \_\_init\_\_(self, position, title, length):
* """Set track position, track title and track length of a new Track Object"""
* # --- Attributes -- #
* self.\_\_position = position
* self.\_\_title = title
* self.\_\_length = length
* # -- Properties -- #
* # Track position
* @property
* **def** position (self):
* **return** self.\_\_position
* @position.setter
* **def** position (self, value):
* self.\_\_position = value
* @property
* **def** title (self):
* **return** self.\_\_title
* @title.setter
* **def** title (self, value):
* self.\_\_title = value
* @property
* **def** length (self):
* **return** self.\_\_length
* @length.setter
* **def** length (self, value):
* self.\_\_length = value
* # -- Methods -- #
* **def** \_\_str\_\_(self):
* """Returns Track details as formatted string"""
* **return** '{:>2}\t{} (by: {})'.format(self.position, self.title, self.length)
* **def** get\_record(self) -> str:
* """Returns: Track record formatted for saving to file"""
* **return** '{},{},{}\n'.format(self.position, self.title, self.length)
* **class** CD:
* """Stores data about a CD / Album:
* properties:
* cd\_id: (int) with CD / Album ID
* cd\_title: (string) with the title of the CD / Album
* cd\_artist: (string) with the artist of the CD / Album
* cd\_tracks: (list) with track objects of the CD / Album
* methods:
* get\_record() -> (str)
* add\_track(track) -> None
* rmv\_track(int) -> None
* get\_tracks() -> (str)
* get\_long\_record() -> (str)
* """

* # -- Constructor -- #
* **def** \_\_init\_\_(self, cd\_id: int, cd\_title: str, cd\_artist: str) -> None:
* """Set ID, Title, Artist and an empty list of tracks of a new CD Object"""
* # -- Attributes -- #
* **try**:
* self.\_\_cd\_id = int(cd\_id)
* self.\_\_cd\_title = str(cd\_title)
* self.\_\_cd\_artist = str(cd\_artist)
* self.\_\_tracks = [] #create empty tracklist for each CD object
* **except** Exception as e:
* **raise** Exception('Error setting initial values:\n' + str(e))
* # -- Properties -- #
* # CD ID
* @property
* **def** cd\_id(self):
* **return** self.\_\_cd\_id
* @cd\_id.setter
* **def** cd\_id(self, value):
* **try**:
* self.\_\_cd\_id = int(value)
* **except** Exception:
* **raise** Exception('ID needs to be Integer')
* # CD title
* @property
* **def** cd\_title(self):
* **return** self.\_\_cd\_title
* @cd\_title.setter
* **def** cd\_title(self, value):
* **try**:
* self.\_\_cd\_title = str(value)
* **except** Exception:
* **raise** Exception('Title needs to be String!')
* # CD artist
* @property
* **def** cd\_artist(self):
* **return** self.\_\_cd\_artist
* @cd\_artist.setter
* **def** cd\_artist(self, value):
* **try**:
* self.\_\_cd\_artist = str(value)
* **except** Exception:
* **raise** Exception('Artist needs to be String!')
* # CD tracks
* @property
* **def** cd\_tracks(self):
* **return** self.\_\_tracks


* # -- Methods -- #
* **def** \_\_str\_\_(self):
* """Returns: CD details as formatted string"""
* **return** '{:>2}\t{} (by: {})'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
* **def** get\_record(self):
* """Returns: CD record formatted for saving to file"""
* **return** '{},{},{}\n'.format(self.cd\_id, self.cd\_title, self.cd\_artist)
* **def** add\_track(self, track: Track) -> None:
* """Adds a track to the CD / Album
* Args:
* track (Track): Track object to be added to CD / Album.
* Returns:
* None.
* """
* self.\_\_tracks.append(track)
* self.\_\_sort\_tracks()

* **def** rmv\_track(self, track\_id: int) -> None:
* """Removes the track identified by track\_id from Album
* Args:
* track\_id (int): ID of track to be removed.
* Returns:
* None.
* """
* **del** self.\_\_tracks[track\_id - 1]
* self.\_\_sort\_tracks()
* **def** \_\_sort\_tracks(self):
* """Sorts the tracks using Track.position. Fills blanks with None"""
* n = len(self.\_\_tracks) #assigning the number of tracks within a CD to a new variable
* **for** track **in** self.\_\_tracks: # for each track stored in the track list of the CD object
* **if** (track **is** **not** None) **and** (n < track.position): # check if the track exists
* n = track.position
* tmp\_tracks = [None] \* n
* **for** track **in** self.\_\_tracks: #for each track listed in the actual track list of the CD object
* **if** track **is** **not** None: # if the track in the track list is already defined THEN
* tmp\_tracks[track.position - 1] = track
* self.\_\_tracks = tmp\_tracks
* **def** get\_tracks(self) -> str:
* """Returns a string list of the tracks saved for the Album
* Raises:
* Exception: If no tracks are saved with album.
* Returns:
* result (string):formatted string of tracks.
* """
* self.\_\_sort\_tracks()
* **if** len(self.\_\_tracks) < 1:
* **raise** Exception('No tracks saved for this Album')
* result = ''
* **for** track **in** self.\_\_tracks:
* **if** track **is** None:
* result += 'No Information for this track\n'
* **else**:
* result += str(track) + '\n'
* **return** result
* **def** get\_long\_record(self) -> str:
* """gets a formatted long record of the Album: Album information plus track details
* Returns:
* result (string): Formatted information about ablum and its tracks.
* """
* result = self.get\_record() + '\n'
* result += self.get\_tracks() + '\n'
* **return** result

ProcessingClasses

* #------------------------------------------#
* # Title: Processing Classes
* # Desc: A Module for processing Classes
* # Change Log: (Who, When, What)
* # DBiesinger, 2030-Jan-01, Created File
* # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
* # shakedbason, 2020-Sep-07,add\_cd and select\_cd
* #------------------------------------------#
* **if** \_\_name\_\_ == '\_\_main\_\_':
* **raise** Exception('This file is not meant to ran by itself')
* **import** DataClasses as DC
* **class** DataProcessor:
* """Processing the data in the application"""
* @staticmethod
* **def** add\_CD(CDInfo, table):
* """function to add CD info in CDinfo to the inventory table.

* Args:
* CDInfo (tuple): Holds information (ID, CD Title, CD Artist) to be added to inventory.
* table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
* Returns:
* None.
* """
* cdId, title, artist = CDInfo
* **try**:
* cdId = int(cdId)
* **except**:
* **raise** Exception('ID must be an Integer!')
* row = DC.CD(cdId, title, artist)
* table.append(row)
* @staticmethod
* **def** select\_cd(table: list, cd\_idx: int) -> DC.CD:
* """selects a CD object out of table that has the ID cd\_idx
* Args:
* table (list): Inventory list of CD objects.
* cd\_idx (int): id of CD object to return
* Raises:
* Exception: If id is not in list.
* Returns:
* row (DC.CD): CD object that matches cd\_idx
* """
* **try**:
* cdID = int(cd\_idx)
* **except**:
* **raise** Exception('NOT A NUMBER!')
* **for** row **in** table:
* **if** row.cd\_id == cdID:
* **print**(f'The following CD has been selected\n{row}\n')
* **return** row
* **raise** Exception('ID IS NOT EXIST')

* @staticmethod
* **def** add\_track(track\_info: tuple, cd: DC.CD) -> None:
* """adds a Track object with attributes in track\_info to cd

* Args:
* track\_info (tuple): Tuple containing track info (position, title, Length).
* cd (DC.CD): cd object the tarck gets added to.
* Raises:
* Exception: DESCraised in case position is not an integer.
* Returns:
* None: DESCRIPTION.
* """
* idPos, title, length = track\_info
* **try**:
* idPos = int(idPos)
* **except** Exception as e:
* **raise** Exception('can not add track:\n' + str(e))
* trackObj = DC.Track(idPos, title, length)
* cd.add\_track(trackObj)

IO classes

* #------------------------------------------#
* # Title: IO Classes
* # Desc: A Module for IO Classes
* # Change Log: (Who, When, What)
* # DBiesinger, 2030-Jan-01, Created File
* # DBiesinger, 2030-Jan-02, Extended functionality to add tracks
* # shakedbason, 2020-Sept-07, Added code save and load
* #------------------------------------------#
* **if** \_\_name\_\_ == '\_\_main\_\_':
* **raise** Exception('This file is not meant to run by itself')
* **import** DataClasses as DC
* **import** ProcessingClasses as PC
* **class** FileIO:
* """Processes data to and from file:
* properties:
* methods:
* save\_inventory(file\_name, lst\_Inventory): -> None
* load\_inventory(file\_name): -> (a list of CD objects)
* """
* @staticmethod
* **def** save\_inventory(file\_name: list, lst\_Inventory: list) -> None:
* """

* Args:
* file\_name (list): list of file names [CD Inventory, Track Inventory] that hold the data.
* lst\_Inventory (list): list of CD objects.
* Returns:
* None.
* """
* file\_name\_CD = file\_name[0]
* file\_name\_track = file\_name[1]
* **try**:
* #Saving CD to file
* with open(file\_name\_CD, 'w') as file:
* **for** disc **in** lst\_Inventory:
* file.write(disc.get\_record())
* #save track data to file
* with open(file\_name\_track, 'w') as file:
* **for** disc **in** lst\_Inventory:
* lstObj = disc.cd\_tracks
* discID = disc.cd\_id
* **for** track **in** lstObj:
* **if** track **is** **not** None:
* strIDplusTrack ='{},{}'.format(discID, track.get\_record())
* file.write(strIDplusTrack)
* # file.write(disc.cd\_id, track.get\_record())
* file.write('{},{}'.format(disc.cd\_id, track.get\_record()))
* **except** Exception as e:
* **print**('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')


* @staticmethod
* **def** load\_inventory(file\_name: list) -> list:
* """

* Args:
* file\_name (list): list of file names [CD Inventory, Track Inventory] that hold the data.
* Returns:
* list: list of CD objects.
* """
* lst\_Inventory = []
* file\_name\_CD = file\_name[0]
* file\_name\_track = file\_name[1]
* **try**:
* with open(file\_name\_CD, 'r') as file:
* **for** line **in** file:
* data = line.strip().split(',')
* row = DC.CD(data[0], data[1], data[2])
* lst\_Inventory.append(row)
* #load track data
* with open(file\_name\_track, 'r') as file:
* **for** line **in** file:
* trackData = line.strip().split(',')
* current = PC.DataProcessor.select\_cd(lst\_Inventory, trackData[0])
* new\_track = DC.Track(int(trackData[1]), trackData[2], trackData[3])
* current.add\_track(new\_track)
* **except** Exception as e:
* **print**('There was a general error!', e, e.\_\_doc\_\_, type(e), sep='\n')
* **return** lst\_Inventory
* **class** ScreenIO:
* """Handling Input / Output"""
* @staticmethod
* **def** print\_menu():
* """Displays a menu of choices to the user
* Args:
* None.
* Returns:
* None.
* """
* **print**('Main Menu\n\n[l] load Inventory from file\n[a] Add CD / Album\n[d] Display Current Inventory')
* **print**('[c] Choose CD / Album\n[s] Save Inventory to file\n[x] exit\n')
* @staticmethod
* **def** menu\_choice():
* """Gets user input for menu selection
* Args:
* None.
* Returns:
* choice (string): a lower case sting of the users input out of the choices l, a, d, c, s or x
* """
* choice = ' '
* **while** choice **not** **in** ['l', 'a', 'd', 'c', 's', 'x']:
* choice = input('Which operation would you like to perform? [l, a, d, c, s or x]: ').lower().strip()
* **print**() # Add extra space for layout
* **return** choice
* @staticmethod
* **def** print\_CD\_menu():
* """Displays a sub menu of choices for CD / Album to the user
* Args:
* None.
* Returns:
* None.
* """
* **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')
* @staticmethod
* **def** menu\_CD\_choice():
* """Gets user input for CD sub menu selection
* Args:
* None.
* Returns:
* choice (string): a lower case sting of the users input out of the choices a, d, r or x
* """
* choice = ' '
* **while** choice **not** **in** ['a', 'd', 'r', 'x']:
* choice = input('Which operation would you like to perform? [a, d, r or x]: ').lower().strip()
* **print**() # Add extra space for layout
* **return** choice
* @staticmethod
* **def** show\_inventory(table):
* """Displays current inventory table

* Args:
* table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
* Returns:
* None.
* """
* **print**('======= The Current Inventory: =======')
* **print**('ID\tCD Title (by: Artist)\n')
* **for** row **in** table:
* **print**(row)
* **print**('======================================')
* @staticmethod
* **def** show\_tracks(cd):
* """Displays the Tracks on a CD / Album
* Args:
* cd (CD): CD object.
* Returns:
* None.
* """
* **print**('====== Current CD / Album: ======')
* **print**(cd)
* **print**('=================================')
* **print**(cd.get\_tracks())
* **print**('=================================')
* @staticmethod
* **def** get\_CD\_info():
* """function to request CD information from User to add CD to inventory

* Returns:
* cdId (string): Holds the ID of the CD dataset.
* cdTitle (string): Holds the title of the CD.
* cdArtist (string): Holds the artist of the CD.
* """
* cdId = input('Enter ID: ').strip()
* cdTitle = input('What is the CD\'s title? ').strip()
* cdArtist = input('What is the Artist\'s name? ').strip()
* **return** cdId, cdTitle, cdArtist
* @staticmethod
* **def** get\_track\_info():
* """function to request Track information from User to add Track to CD / Album

* Returns:
* trkId (string): Holds the ID of the Track dataset.
* trkTitle (string): Holds the title of the Track.
* trkLength (string): Holds the length (time) of the Track.
* """
* trkId = input('Enter Position on CD / Album: ').strip()
* trkTitle = input('What is the Track\'s title? ').strip()
* trkLength = input('What is the Track\'s length? ').strip()
* **return** trkId, trkTitle, trkLength

TestHarness

* #------------------------------------------#
* # Title: Test Harness
* # Desc: A Module to test the Modules
* # Change Log: (Who, When, What)
* # DBiesinger, 2030-Jan-01, Created File
* # DBiesinger, 2030-Jan-02, Extended functionality to add tracks

* **import** DataClasses as DC
* **import** ProcessingClasses as PC
* **import** IOClasses as IO
* lstOfCDObjects = []
* file\_name = ['TestCD.txt', 'TestTrack.txt']
* **print**('\n\nTesting Track class')
* **print**(DC.Track.\_\_doc\_\_)
* trk1 = DC.Track(1, 'test.track1', '01:59')
* trk2 = DC.Track(2, 'test.track2', '02:59')
* **print**(trk1)
* **print**('record for file:', trk1.get\_record())
* **print**('\n\nTesting CD class')
* **print**(DC.CD.\_\_doc\_\_)
* cd1 = DC.CD(1, 'test\_title', 'cd\_artist')
* **print**(cd1)
* **print**('record for file:', cd1.get\_record())
* **print**('adding tracks...')
* cd1.add\_track(trk1)
* cd1.add\_track(trk2)
* **print**('get tracks:\n', cd1.get\_tracks())
* **print**('get long record:\n', cd1.get\_long\_record())
* **print**('removing track 2...')
* cd1.rmv\_track(2)
* **print**('get long record:\n', cd1.get\_long\_record())
* lstOfCDObjects.append(cd1)
* **print**('\n\nTesting of class FileIO')
* IO.FileIO.save\_inventory(file\_name, lstOfCDObjects)
* **print**(IO.FileIO.load\_inventory(file\_name))
* **print**('\n\nTesting ScreenIO class')
* **print**('Main menu:')
* IO.ScreenIO.print\_menu()
* **print**('selection in menu: {}'.format(IO.ScreenIO.menu\_choice()))
* **print**('Inventory:')
* IO.ScreenIO.show\_inventory(lstOfCDObjects)
* cd2 = DC.CD(2, 'test\_title\_2', 'cd\_artist\_2')
* lstOfCDObjects.append(cd2)
* **print**('Inventory:')
* **for** item **in** lstOfCDObjects:
* **print**(item)
* cd\_idx = 1
* cd = PC.DataProcessor.select\_cd(lstOfCDObjects, cd\_idx)
* **print**('\nSub Menu')
* IO.ScreenIO.print\_CD\_menu()
* **print**('selection in sub menu: {}'.format(IO.ScreenIO.menu\_CD\_choice()))
* **print**('Tracks:')
* IO.ScreenIO.show\_tracks(cd)
* **print**('\n\nTesting Processing Classes')
* PC.DataProcessor.add\_CD((3, 'Foreigner', 'Foreigner'), lstOfCDObjects)
* **print**('Inventory:')
* **for** item **in** lstOfCDObjects:
* **print**(item)