

Functions and Classes

Introduction

Returning to the CD Inventory program, this assignment will focus on using the program to showcase optimal separation of concerns by using functions and classes to organize code blocks and data flow. The assignment will hence examine the various components of functions, and how variables interact with them.

Overview of Functions

Functions are widely useful in programming for improving code's organization and efficiency. But for all the benefits they offer, they are conceptually quite simple. At heart, a function is a tool that is used to make a block of code statements identifiable as a distinct group that can then be called and executed in a single instruction using a name created by the programmer. In Python, the function must be defined before it is called, meaning the relevant block of code must be written first in the order of the program's instructions before the function's name is invoked.¹

```
170.     @staticmethod
171.     def add_cd_input(table, IDchecklist):

...

184.         ID = ' '
185.         rowcount = int()
186.         while rowcount < len(table):
187.             ID = input('Enter numerical ID: ').strip()
188.             if ID.isnumeric() is True:
189.                 for row in table:
190.                     if row['ID'] == ID:
191.                         print('That ID already exists. Please enter a new ID.')
192.                         rowcount = 0
193.                 else:
194.                     rowcount += 1
195.             IDchecklistitemcount = int()
196.             while IDchecklistitemcount < len(IDchecklist):
197.                 for item in IDchecklist:
198.                     if item == int(ID):
199.                         print('That ID already exists. Please enter a new ID.')
200.                         IDchecklistitemcount = 0
201.                 else:
202.                     IDchecklistitemcount += 1
203.             else:
204.                 continue
205.             IDchecklist.append(ID)
206.             title = input('What is the CD\'s title? ').strip()
207.             artist = input('What is the Artist\'s name? ').strip()
208.             return IDchecklist, ID, title, artist
```

Listing 1 - Excerpt, Appendix A: Function for prompting user input for a new CD entry

¹ "Module 06 of 10", *Foundations of Programming (Python)*, Dirk Biesinger, p2 (accessed August 16, 2020).

Above is an example of a robust I/O function called `add_cd_input()` which contains which requires and produces several different data values during its execution. The prerequisite values that the function needs are identified at the same time as the function's name is defined, on line 171, within the brackets: "table", and "IDchecklist". These two terms are not the actual data itself, but rather parameters, which are essentially variables created by and for the function during its execution.² Both table and IDchecklist are referred to repeatedly within the body of the function from lines 186 to 205 indicating places where the data values provided to these parameters would be used.

The very last line of the function, 208, is initiated with the keyword "return". This indicates that the list of variables following it, "IDchecklist", "ID", "title", and "artist" will be used by the function to generate return values which can then be assigned to new variables outside the scope of the function and used for other purposes. These return values are one of the main uses for functions in general, though they are not a requirement of functions, nor are return values contingent on a function having defined parameters.

Having said that, in the case of `add_cd_input()`, two of the variables used to generate return values, IDchecklist and ID, rely on a series of instructions which are executed using information from the function's parameters. Note that IDchecklist is employed as both a parameter and a return value in this function! This may seem counterintuitive but can be clarified by revealing where parameters derive their values from.

```
249.         # 3.3 process add a CD
250.         elif menuChoice == 'a':
251.             print("Please provide new CD info.")
252.             cdIDlist = []
253.             add_verification = 'y'
254.             while add_verification == 'y':
255.                 # 3.3.1 Ask user for new ID, CD Title and Artist
256.                 cdIDlist,cdID,cdTitle,cdArtist = IO.add_cd_input(lstTbl, cdIDlist)
257.                 # 3.3.2 Add item to the table and ask user if they want to add another CD
258.                 DataProcessor.add_cd(cdID, cdTitle, cdArtist, lstTbl)
259.                 add_verification = input('Would you like to add another CD? [y/n] ').lower()
260.                 if add_verification == 'n': break
261.             IO.show_inventory(lstTbl)
262.             continue # start loop back at top.
```

Listing 2 - Excerpt, Appendix A: Function call of `add_cd_input()` within the main code block of the CD Inventory program

In Listing 2, `add_cd_input()` is called on line 256. Notice that the two variables contained in the function call's brackets are not the same as the parameter names contained in the function's definition. These two variables are examples of arguments, which a function call uses to pass data values into the function itself. The order the arguments are written in corresponds to the order in which the parameters were defined on line 171 in Listing 1, such that the parameter table acquires the value of `lstTbl` and IDchecklist acquires the value of `cdIDlist`. Thus, where a parameter can be understood as an object for containing external data passed to a function, an argument can be understood as the means of passing that data to the function. While arguments can be used to pass specific values to a function, in general they will be used to pass variables referring to values simply because they are more versatile, typically for repeated use as is the case in Listing 1.³

Line 256 does not only execute a function call, however. The function call is also used to assign values to four variables: `cdIDlist`, `cdID`, `cdTitle`, and `cdArtist`. What is happening here is that these four variables are acquiring the return values passed out through line 208 of the function in Listing 1, in the same order they are written. Hence `cdIDlist` acquires the value of IDchecklist, `cdID` acquires ID, `cdTitle` acquires title, and `cdArtist` acquires artist. It is important to acknowledge that these two sets of variables, from line 256 outside the function and from line 208 inside the function, have very similar corresponding names—in fact, they could share identical names. The reason they do not is also the reason that the values need to be passed from one set of variables to the other in the first place: the return values are moving from

² "Chapter 6 Functions: The Tic-Tac-Toe Game", Python Programming for the Absolute Beginner Third Edition, Michael Dawson, p163.

³ "Module 06 of 10", Dirk Biesinger, p5 (accessed August 16, 2020).

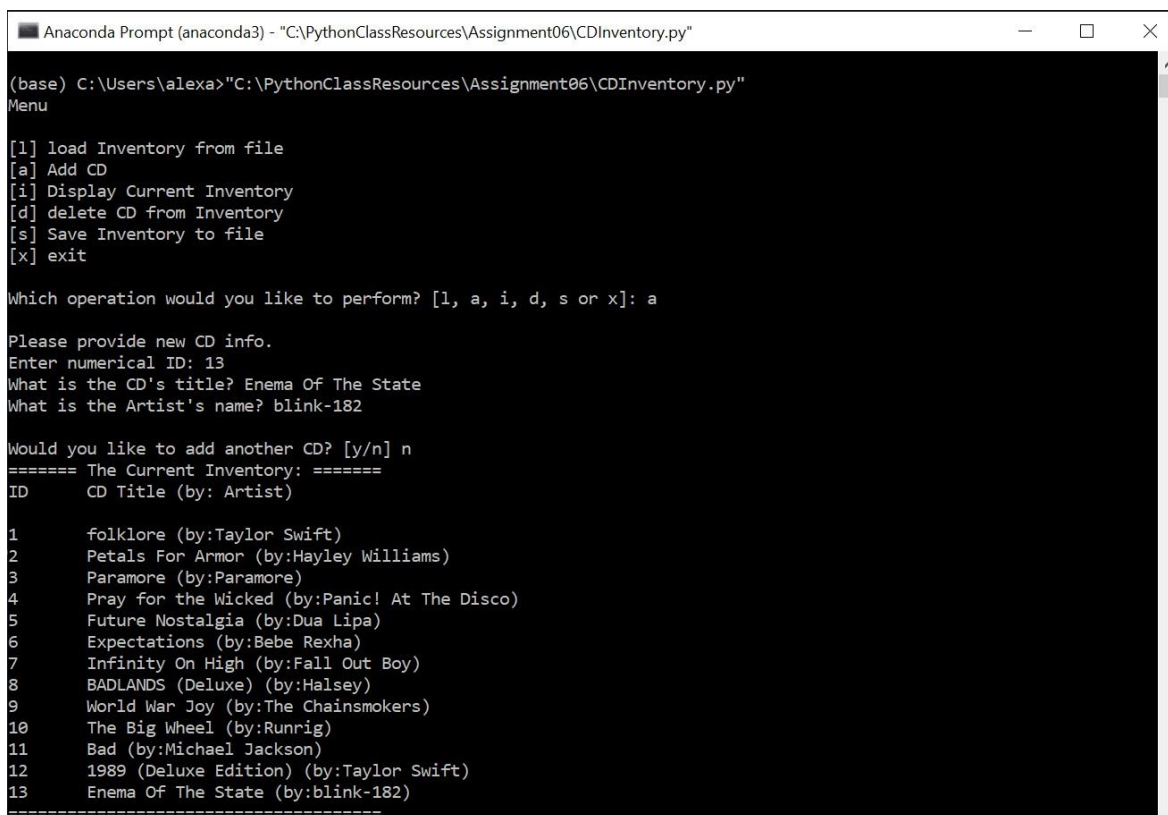
local variables, which are only operative within the bounds of the function, to global variables, which are operative in the main body of the program and can be used elsewhere.

Sidebar: Shadowing Global Variables

Although Python permits programmers to designate separate local and global variables which share exactly the same name, it is inadvisable to do so. This practice, called shadowing, can be confusing even for the original programmer, let alone others who may interact with their code at a later time, because it may not be apparent that there are actually two variables working in different contexts. This confusion could lead to changes being made to the code which cause it to break in an unclear way.⁴

It is for this same reason that IDchecklist is both a parameter and a return value for add_cd_input(). Local variables are reinitialized every time a function is called meaning that they do not hold values generated by previous executions. Because the value generated by IDchecklist is particularly important and needs to persist beyond a single execution of the add_cd_input() function, it is passed out as a return value that can be reapplied to the next execution as an argument containing the previous data in the form of cdIDlist.

On the other hand, the other three return values are more conventionally applied on the following line of code as arguments for an entirely different function, add_cd(). Notice that add_cd_input() and add_cd(), while similarly named, have different prefixes: "IO" for the former, and "DataProcessor" for the latter. This is because they belong to different classes of functions. Classes, like functions, serve an organizational purpose, and are programmer-defined. The only difference is that where functions create groups of code statements, classes are used to group functions.⁵ And indeed, because both the functions and the classes in Listing 2 are both clearly named, their connected purpose may be self-evident: IO.add_cd_input() generates user data input, and DataProcessor.add_cd() processes that user data for later storage in the CD Inventory text file. Their combined operation is demonstrated below in Figure 1.



```
Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Assignment06\CDInventory.py"
(base) C:\Users\alexa>"C:\PythonClassResources\Assignment06\CDInventory.py"
Menu
[1] load Inventory from file
[a] Add CD
[i] Display Current Inventory
[d] delete CD from Inventory
[s] Save Inventory to file
[x] exit

Which operation would you like to perform? [1, a, i, d, s or x]: a

Please provide new CD info.
Enter numerical ID: 13
What is the CD's title? Enema Of The State
What is the Artist's name? blink-182

Would you like to add another CD? [y/n] n
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       folklore (by:Taylor Swift)
2       Petals For Armor (by:Hayley Williams)
3       Paramore (by:Paramore)
4       Pray for the Wicked (by:Panic! At The Disco)
5       Future Nostalgia (by:Dua Lipa)
6       Expectations (by:Bebe Rexha)
7       Infinity On High (by:Fall Out Boy)
8       BADLANDS (Deluxe) (by:Halsey)
9       World War Joy (by:The Chainsmokers)
10      The Big Wheel (by:Runrig)
11      Bad (by:Michael Jackson)
12      1989 (Deluxe Edition) (by:Taylor Swift)
13      Enema Of The State (by:blink-182)
=====
```

Figure 1 - Execution of IO.add_cd_input() and DataProcessor.add_cd()

⁴ Ibid, p17 (accessed August 16, 2020).

⁵ Ibid, p21 (accessed August 16, 2020).

Sidebar: How Functions Address Separation of Concerns.

As described above, functions can be grouped together in discrete classes corresponding to the type of operation they perform during execution of the program. Most commonly, per separation of concerns best practices, these operations are divided into I/O and processing, just as the two functions `IO.add_cd_input()` and `DataProcessor.add_cd` are. In other words, functions are written and physically organized in a program's code in a way such that I/O and processing components of the program are neatly separated, with the main code block containing all the function calls following them both.

Summary

At the outset, the challenges posed by this assignment were somewhat confusing to me, simply because I was not yet used to the mindset of thinking in functions and how data flows between them. However, once I began building my first functions with the pre-existing code, I quickly became used to this. I am especially pleased with the complex data relationships I developed for `add_cd_input()`.

Appendix

A – CD Inventory Code

```
11. # -- DATA -- #
12. menuChoice = '' # User input
13. lstTbl = [] # list of lists to hold data
14. dicRow = {} # list of data row
15. strFileName = 'CDInventory.txt' # data storage file
16. objFile = None # file object
17.
18.
19. # -- PROCESSING -- #
20. class DataProcessor:
21.
22.     @staticmethod
23.     def add_cd(ID, title, artist, table):
24.         """Collects new CD data from user and converts into dict for appending to current inventory t
25.         able
26.
27.         Args:
28.             ID (string): numerical identification for the new CD
29.             title (string): album title of the new CD
30.             artist (string): artist name of the new CD
31.             table (list of dicts): 2D data structure (list of dicts) that holds the data during runti
32.         me
33.
34.         Returns:
35.             None.
36.         """
37.         dicRow = {'ID': ID, 'Title': title, 'Artist': artist}
38.         table.append(dicRow)
39.         print()
40.
41.     @staticmethod
42.     def delete_cd(delID, table):
43.         """Deletes dicts by ID key from 2D data structure
44.
45.         Args:
46.             delID (list of strings): holds one or more ID values designated for deleiton
47.             table (list of dicts): 2D data structure (list of dicts) that holds the data during runti
48.         me
49.
50.         Returns:
51.             None.
52.         """
53.         for row in delID: delID = row.strip().split(',')
54.
```

```

51.         delIDln = int(len(delID))
52.         cdcount = int()
53.         while cdcount < delIDln: # this loop causes an interminable error if delID item is not in tab
le
54.             for item in delID:
55.                 rowcount = int()
56.                 while rowcount < len(table):
57.                     for row in table:
58.                         if row['ID'] == item:
59.                             rowcount += 1
60.                             cdcount += 1
61.                             table.remove(row)
62.                             delID.remove(item)
63.                             break
64.                         else:
65.                             rowcount += 1
66.                             continue
67.
68.             if len(delID) == 0: print('IDs deleted.')
69.             elif cdcount == 0: print('IDs not found.')
70.             elif cdcount != len(delID) and len(delID) > 0: print('One or more IDs was not found:',delID[:
])
71.
72. class FileProcessor:
73.     """Processing the data to and from text file"""
74.
75.     @staticmethod
76.     def read_file(file_name, table):
77.         """Function to manage data ingestion from file to a list of dictionaries
78.
79.         Reads the data from file identified by file_name into a 2D table
80.         (list of dicts) table one line in the file represents one dictionary row in table.
81.
82.         Args:
83.             file_name (string): name of file used to read the data from
84.             table (list of dict): 2D data structure (list of dicts) that holds the data during runtim
e
85.
86.         Returns:
87.             None.
88.         """
89.         table.clear() # this clears existing data and allows to load data from file
90.         objFile = open(file_name, 'r')
91.         for line in objFile:
92.             data = line.strip().split(',')
93.             dicRow = {'ID': data[0], 'Title': data[1], 'Artist': data[2]}
94.             table.append(dicRow)
95.         objFile.close()
96.
97.     @staticmethod
98.     def write_file(file_name, table):
99.         """Function to manage transcription of data from list of dictionaries in
100.         current inventory to file
101.
102.         Args:
103.             file_name (string): name of file used to wrtie the data to
104.             table (list of dict): 2D data structure (list of dicts) that holds the data during
runtime
105.
106.         Returns:
107.             None.
108.         """
109.         objFile = open(file_name, 'w')
110.         for row in table:
111.             lstValues = list(row.values())
112.             lstValues[0] = str(lstValues[0])
113.             objFile.write(','.join(lstValues) + '\n')
114.         objFile.close()

```

```

115.
116.     # -- PRESENTATION (Input/Output) -- #
117.
118.     class IO:
119.         """Handling Input / Output"""
120.
121.         @staticmethod
122.         def print_menu():
123.             """Displays a menu of choices to the user
124.
125.             Args:
126.                 None.
127.
128.             Returns:
129.                 None.
130.             """
131.
132.             print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory
133.             print('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
134.
135.         @staticmethod
136.         def menu_choice():
137.             """Gets user input for menu selection
138.
139.             Args:
140.                 None.
141.
142.             Returns:
143.                 choice (string): a lower case sting of the users input out of the choices l, a, i,
144.                 d, s or x
145.             """
146.             choice = ' '
147.             while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
148.                 choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: '
149.                 ).lower().strip()
150.                 print() # Add extra space for layout
151.                 return choice
152.
153.         @staticmethod
154.         def show_inventory(table):
155.             """Displays current inventory table
156.
157.             Args:
158.                 table (list of dict): 2D data structure (list of dicts) that holds the data during
159.                 runtime.
160.
161.             Returns:
162.                 None.
163.             """
164.             print('==== The Current Inventory: =====')
165.             print('ID\tCD Title (by: Artist)\n')
166.             for row in table:
167.                 print('{ }\t{ } (by: { })'.format(*row.values()))
168.             print('=====')
169.
170.         @staticmethod
171.         def add_cd_input(table, IDchecklist):
172.             """Gets user input for new CD information
173.
174.             Args:
175.                 table (list of dict): 2D data structure (list of dicts) that holds the data during
176.                 runtime.
177.                 IDchecklist (list): persistent list of new IDs being added to current inventory to
178.                 prevent accidental duplicate input by user

```

```

177.
178.         Returns:
179.             IDchecklist (list): persistent list of new IDs being added to current inventory to
prevent accidental duplicate input by user
180.             ID (string): numerical identification for the new CD
181.             title (string): album title of the new CD
182.             artist (string): artist name of the new CD
183.         """
184.         ID = ' '
185.         rowcount = int()
186.         while rowcount < len(table):
187.             ID = input('Enter numerical ID: ').strip()
188.             if ID.isnumeric() is True:
189.                 for row in table:
190.                     if row['ID'] == ID:
191.                         print('That ID already exists. Please enter a new ID.')
192.                         rowcount = 0
193.                     else:
194.                         rowcount += 1
195.                 IDchecklistitemcount = int()
196.                 while IDchecklistitemcount < len(IDchecklist):
197.                     for item in IDchecklist:
198.                         if item == int(ID):
199.                             print('That ID already exists. Please enter a new ID.')
200.                             IDchecklistitemcount = 0
201.                         else:
202.                             IDchecklistitemcount += 1
203.                 else:
204.                     continue
205.                 IDchecklist.append(ID)
206.                 title = input('What is the CD\'s title? ').strip()
207.                 artist = input('What is the Artist\'s name? ').strip()
208.                 return IDchecklist, ID, title, artist
209.
210.     # 1. When program starts, read in the currently saved Inventory
211.     FileProcessor.read_file(strFileName, lstTbl)
212.
213.     # 2. start main loop
214.     while True:
215.         # 2.1 Display Menu to user and get choice
216.         IO.print_menu()
217.         menuChoice = IO.menu_choice()
218.
219.         # 3. Process menu selection
220.         # 3.1 process exit first
221.         if menuChoice == 'x':
222.             break
223.         # 3.2 process load inventory
224.         if menuChoice == 'l':
225.             print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-
loaded from file.')
226.             overwrite_verification = input('Type \'yes\' to continue and reload from file. Otherwi
se reload will be canceled.\n')
227.             if overwrite_verification.lower() == 'yes':
228.                 print('reloading...')
229.                 FileProcessor.read_file(strFileName, lstTbl)
230.                 IO.show_inventory(lstTbl)
231.             else:
232.                 input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the
menu.')
233.                 IO.show_inventory(lstTbl)
234.                 continue # start loop back at top.
235.         # 3.3 process add a CD
236.         elif menuChoice == 'a':
237.             print("Please provide new CD info.")
238.             cdIDlist = []
239.             add_verification = 'y'
240.             while add_verification == 'y':

```

```

241.         # 3.3.1 Ask user for new ID, CD Title and Artist
242.         cdIDlist,cdID,cdTitle,cdArtist = IO.add_cd_input(lstTbl, cdIDlist)
243.         # 3.3.2 Add item to the table and ask user if they want to add another CD
244.         DataProcessor.add_cd(cdID, cdTitle, cdArtist, lstTbl)
245.         add_verification = input('Would you like to add another CD? [y/n] ').lower()
246.         if add_verification == 'n': break
247.         IO.show_inventory(lstTbl)
248.         continue # start loop back at top.
249.     # 3.4 process display current inventory
250.     elif menuChoice == 'i':
251.         IO.show_inventory(lstTbl)
252.         continue # start loop back at top.
253.     # 3.5 process delete a CD
254.     elif menuChoice == 'd':
255.         # 3.5.1 get Userinput for which CD to delete
256.         # 3.5.1.1 display Inventory to user
257.         IO.show_inventory(lstTbl)
258.         # 3.5.1.2 ask user which ID to remove
259.         cdIDdel = [input('Enter one or more IDs to delete, separated by commas (example: "1,2,
3"): ').strip()]
260.         # 3.5.2 search through table and delete CD
261.         DataProcessor.delete_cd(cdIDdel, lstTbl)
262.         # 3.5.3 display altered Inventory to user
263.         IO.show_inventory(lstTbl)
264.         continue # start loop back at top.
265.     # 3.6 process save inventory to file
266.     elif menuChoice == 's':
267.         # 3.6.1 Display current inventory and ask user for confirmation to save
268.         IO.show_inventory(lstTbl)
269.         save_verification = input('Save this inventory to file? [y/n] ').strip().lower()
270.         # 3.6.2 Process choice
271.         if save_verification == 'y':
272.             # 3.6.2.1 save data
273.             FileProcessor.write_file(strFileName, lstTbl)
274.         else:
275.             input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
276.         continue # start loop back at top.
277.     # 3.7 catch-
278.     all should not be possible, as user choice gets vetted in IO, but to be safe:
279.     else:
280.         print('General Error')

```

B – Github Link

https://github.com/Alex-Angelico/Assignment_06