

Jeffrey Shen

3/8/2020

Foundations of Programming (Python)

Assignment 07

Modules and Structured Error Handling with Python

Introduction

In this document, I will provide an overview of using external Python modules and structured error handling. This report will include discussion of the CDInventory_07.py script and what challenges I came across.

CD Inventory Script

Using Pickle Module

Building upon the last few week's assignments of managing a CD inventory based on user input, I integrated a module and structured error handling. I will only describe the changes included to the script, since majority of the pseudocode and functionality stayed the same.

[l] load Inventory from file	if strChoice == 'x'
[a] Add CD	do this
[i] Display Current Inventory	elif strChoice == 'l'
[d] delete CD from Inventory	do this
[s] Save Inventory to file	elif strChoice == 'a'
[x] exit	do this
	elif strChoice == 'i'
	do this
	elif strChoice == 'd'
	do this
	elif strChoice == 's'
	do this

I used the pickle module for this assignment. Pickle implements binary protocols for serializing and de-serializing Python object structure.¹ Python is able to interpret and convert this information between byte stream and object hierarchy. To use the pickle module, I simple used the syntax: "import pickle" There is documentation on how to use pickle since it has certain attributes (e.g. pickle.load, pickle.dump)² Here is a snippet of my code utilizing pickle for file operations.

¹ <https://docs.python.org/3/library/pickle.html>

² <https://wiki.python.org/moin/UsingPickle>

```

81 .....# try to read file name from binary
82 .....# if there is an error, FileNotFoundError is returned
83 .....try:
84 .....    with open(file_name, 'rb') as objFile:
85 .....        table = pickle.load(objFile)
86 .....    except FileNotFoundError:
87 .....        print('File not found')
88 .....    return table
89
90 .....@staticmethod
91 .....def write_file(file_name, table):
92 .....    """Writes file data using pickle module
93
94 .....    Args:
95 .....        file_name (string): name of file used to read the data from
96 .....        table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
97
98 .....    Returns:
99 .....        None.
100 .....    """
101 .....    # write the file in binary
102 .....    with open(file_name, 'wb') as objFile:
103 .....        pickle.dump(table, objFile)
104

```

Figure 1 Pickle Module

The try-except error handling structure will be discussed later in this report. I will focus on the pickle syntax for now. I used the same “with open...” syntax to call out what file I want to operate on. However, in this case, in line 84 I use ‘rb’ which stands for read binary. This tells Python that this is not a text file. I had a lot of difficulty understanding and implementing this. I kept on trying to use pickle to read text file. The table is populated with the data inside the file by using “pickle.load(objFile)” in line 85. The same type of syntax is used to save a file but in this case, I use ‘wb’ which stands for write binary. This tells Python to write in binary format, and I call out “pickle.dump” to use the pickle module. Using pickle simplifies the arguments and parameters we need to pass through as it is inherently part of the Python module.

Structured Error Handling

The idea behind this is to learn how to handle errors in my Python code. There can be different types of errors: ZeroDivisionError, TypeError, ValueError, FileNotFoundError, Exception, and more. The general syntax is to use a try-except format.

```

130 .....@staticmethod
131 .....def get_user_input():
132 .....    """Gets ID, Artist, and Album information from the user
133
134 .....    Args:
135 .....        None.
136
137 .....    Return:
138 .....        cd_id (int): integer representing the ID of the album
139 .....        title (string): string representing the Title of the album
140 .....        artist (string): string representing the Artist
141 .....    """
142 .....    # continue to loop if user input is returning an error
143 .....    while True:
144 .....        try:
145 .....            cd_id = int(input('Enter ID: '))
146 .....            title = input('What is the CD\'s title? ').strip()
147 .....            artist = input('What is the Artist\'s name? ').strip()
148 .....            return cd_id, title, artist
149 .....        except ValueError as e:
150 .....            print('Not an integer')
151 .....            print('Build in error info:')
152 .....            print(type(e), e, e.__doc__, sep='\n')
153 .....            print() # extra space for layout
154 .....            IO.show_inventory(lstTbl)
155

```

Figure 2 Try-Except

For example, in my get_user_input() function, line 144 calls out “try:” Following in lines 145-148 is the parameters that I would like to receive from the user. However, if there is an error passed (in this case if cd_id is not an integer), the program will proceed to line 149 except clause. Lines 150-153 will output what the error code was and how the user can proceed. I included line 153-154 for formatting and

helping remind the user what is in the table. At first, I included multiple try-except structures to cycle through, then realized, the user can indefinitely make wrong inputs so I created a while loop.

```
105 '''PRESENTATION'''
106 class IO:
107     """Processing I/O operations"""
108     @staticmethod
109     def del_input():
110         """Gets ID that user wants to delete.
111
112         Args:
113             None.
114
115         Return:
116             strIDDel.
117         """
118         # continue to loop if user input is returning an error
119         while True:
120             try:
121                 strIDDel = int(input('Which ID would you like to delete? ').strip())
122                 return strIDDel
123             except ValueError as e:
124                 print('Not an integer')
125                 print('Build in error info:')
126                 print(type(e), e, e.__doc__, sep='\n')
127                 print() # extra space for layout
128                 IO.show_inventory(lstTbl)
```

Figure 3 del_input() function

The same idea is implemented here in my function del_input(). The user needs to input an integer for ID or there will be an except clause that outputs error messages. I actually included this del_input() function as I didn't create it from last week. I noticed that it could be included in my IO class. In line 246, the function is called out.

```
240 ...# 3.5 process delete a CD
241 ...elif strChoice == 'd':
242     ...# 3.5.1 get User input for which CD to delete
243     ...# 3.5.1.1 display Inventory to user
244     ...IO.show_inventory(lstTbl)
245     ...# 3.5.1.2 ask user which ID to remove
246     ...strIDDel = IO.del_input()
247     ...# 3.5.2 search thru table and delete CD
248     ...lstTbl = DataProcessor.user_del(strIDDel, lstTbl)
249     ...# show updated table
250     ...IO.show_inventory(lstTbl)
251     ...continue # start loop back at top.
252
```

Figure 4 Calling out del_input function

Another change I included from last week was how the user add inputs from the main while loop. Originally, in line 231 was "lstTbl = DataProcessor.user_add(*IO.get_user_input(), lstTbl)" however, I kept on getting errors on non-iterable values, if the IO.get_user_input() pushed an error to the user. I broke it out to two lines so I could manage that better.

```
226 ...# 3.3 process add a CD
227 ...elif strChoice == 'a':
228     ...# 3.3.1 Ask user for new ID, CD Title and Artist
229     ...# 3.3.2 Add item to the table
230     ...cd_id, title, artist = IO.get_user_input()
231     ...lstTbl = DataProcessor.user_add(cd_id, title, artist, lstTbl)
232     ...IO.show_inventory(lstTbl)
233     ...continue # start loop back at top.
```

Figure 5 "Adding" function calls

Example runs from Spyder and the terminal are included in the Appendix.

Questions

- What are the benefits of using structured error handling?
 - The error messages can be integrated into the program and allow better readability/debugging. The idea of structured error handling helps programmer better understands the error messages the Python retrieves.
- What are the differences between a text file and a binary file?
 - A text file is readable to humans (generally, or that is the assumption). Generally including numbers, strings. A binary file is what a sequence of bytes (binary digits, bits). A computer can read binary formats and interpret it back to the user. Using a binary reader is needed to understand.
- How is the Exception class used?
 - Exception classes are utilized for error handling. They can be implemented and called out using "raise" which the program will then call out the exception class.
- How do you "derive" a new class from the Exception class?
 - Derived classes are inherited from the base class. They can be customized to meet specific needs.
- When might you create a class derived from the Exception class?
 - If there are specific exceptions, you want to call out or help the program go back to a "Traceback"
- What is the Markdown language?
 - It's a way of organizing the readme.md file to allow better understanding and consistency across programmers to view what is included in a repository.

Summary

In this lab, I explored using modules and structured error handling to enhance last week's assignment. The overall functionality stayed the same for managing the CD inventory but included best practices. In the beginning, I struggled to understand and digest how to use the pickle module. I could not integrate saving and loading accurately. Comprehending the syntax and using module attributes proved to be a more challenging task than I thought. Structured error handling was a much easier concept to grasp as it was similar to if-else syntax but instead used try-except clauses.

Appendix

Complete Code for CDInventory_07.py

```
1 |
2 |"""CDInventory_07.py
3 |Basic working with classes, functions structured error handling, and pickle module
4 |Dalesinger, 2020-Jan-01, Created File.
5 |Jeffrey Shen, 2020-Feb-25, Edited file for assignments and comments.
6 |Douglas Xiao, 2020-Mar-01, Drafting, revised functions, added parameters.
7 |Jeffrey Shen, 2020-Mar-07, Modified file from Doug's feedback and then added preliminary structure.
8 |Jeffrey Shen, 2020-Mar-08, Included structured error handling and store via binary data (pickle module).
9 |
10 |
11 |"""DATA"""
12 |# Import modules
13 |import pickle
14 |# Initialize variables
15 |strChoice = "" # user input
16 |lstTbl = [] # list of lists to hold data
17 |dicrow = {} # list of data row
18 |strFilename = "CDInventory.dat" # data storage file
19 |objFile = None # file object
20 |
21 |"""Processing"""
22 |class DataProcessor:
23 |    """Processing user action"""
24 |    @staticmethod
25 |    def user_add_cd_id, title, artist, table):
26 |        """Add CD title and artist from user input
27 |
28 |        Args:
29 |            cd_id (string): string representing the ID of the album
30 |            title (string): string representing the title of the album
31 |            artist (string): string representing the artist
32 |            table (list of dicts): 2d structure, list of dictionaries containing cd information
33 |
34 |        Returns:
35 |            table (list of dicts): 2d structure, list of dictionaries containing cd information
36 |
37 |        """
38 |        dicrow = {"ID": cd_id, "Title": title, "Artist": artist}
39 |        table.append(dicrow)
40 |        return table
41 |
42 |    @staticmethod
43 |    def user_del(id_to_delete, table):
44 |        """Deletes ID from user input
45 |
46 |        Args:
47 |            id_to_delete (string): id representing the cd to remove from inventory
48 |            table (list of dicts): 2d structure, list of dictionaries containing cd information
49 |
50 |        Returns:
51 |            table (list of dicts): 2d structure, list of dictionaries containing cd information
52 |
53 |        """
54 |        inRow = -1
55 |        binCDremoved = False
56 |        for row in table:
57 |            if row["ID"] == id_to_delete:
58 |                del table[inRow]
59 |                binCDremoved = True
60 |            inRow += 1
61 |        if binCDremoved:
62 |            print("The CD was removed")
63 |        else:
64 |            print("Could not find this CD")
65 |        return table
66 |
67 |class FileProcessor:
68 |    """Processing file operations"""
69 |    @staticmethod
70 |    def read_file(file_name, table):
71 |        """Function to remove data ingestion from file to a list of dictionaries
72 |
73 |        Reads the data from file identified by file_name using pickle module (binary)
74 |
75 |        Args:
76 |            file_name (string): name of file used to read the data from
77 |            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
78 |
79 |        Returns:
80 |            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
81 |
82 |        """
83 |        # try to read file name from binary
84 |        # if there is an error, FileNotFoundError is returned
85 |        try:
86 |            with open(file_name, "rb") as objFile:
87 |                table = pickle.load(objFile)
88 |        except FileNotFoundError:
89 |            print("File not found")
90 |        return table
91 |
92 |    @staticmethod
93 |    def write_file(file_name, table):
94 |        """Write file data using pickle module
95 |
96 |        Args:
97 |            file_name (string): name of file used to read the data from
98 |            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
99 |
100 |        Returns:
101 |            None
102 |
103 |        """
104 |        # write the file in binary
105 |        with open(file_name, "wb") as objFile:
106 |            pickle.dump(table, objFile)
107 |
108 |"""PRESENTATION"""
109 |class IO:
110 |    """Processing I/O operations"""
111 |    @staticmethod
112 |    def del_input():
113 |        """Gets ID that user wants to delete
114 |
115 |        Args:
116 |            None
117 |
118 |        Returns:
119 |            strIDdel: string representing the ID of the album
120 |
121 |        """
122 |        strIDdel = input("Which ID would you like to delete? ").strip()
123 |        try:
124 |            int(strIDdel)
125 |        except ValueError as e:
126 |            print("Must be an integer")
127 |            print("Build in error info:")
128 |            print(e)
129 |            print("Extra space for layout")
130 |            IO.show_inventory(lstTbl)
131 |        return strIDdel
132 |
133 |    @staticmethod
134 |    def get_user_input():
135 |        """Gets ID, Artist, and Album information from the user
136 |
137 |        Args:
138 |            None
139 |
140 |        Returns:
141 |            cd_id (int): integer representing the ID of the album
142 |            title (string): string representing the title of the album
143 |            artist (string): string representing the artist
144 |
145 |        """
146 |        cd_id = int(input("Enter ID: "))
147 |        title = input("What is the CD's title? ").strip()
148 |        artist = input("What is the artist's name? ").strip()
149 |        return cd_id, title, artist
150 |
151 |    @staticmethod
152 |    def print_menu():
153 |        """Displays a menu of choices to the user
154 |
155 |        Args:
156 |            None
157 |
158 |        Returns:
159 |            None
160 |
161 |        """
162 |        print("""
163 |        1. Add new CD to inventory
164 |        2. Delete CD from inventory
165 |        3. Show current inventory
166 |        4. Save inventory to file
167 |        5. Load inventory from file
168 |        6. Quit
169 |        """)
170 |
171 |    @staticmethod
172 |    def show_inventory(table):
173 |        """Displays current inventory table
174 |
175 |        Args:
176 |            table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
177 |
178 |        Returns:
179 |            None
180 |
181 |        """
182 |        print("\n===== The Current Inventory =====")
183 |        print("ID\tTitle\tArtist\n")
184 |        for row in table:
185 |            print(f"{row['ID']}\t{row['Title']}\t{row['Artist']}")
186 |
187 |    @staticmethod
188 |    def save_data(strFilename, lstTbl):
189 |        """Save data to file
190 |
191 |        Args:
192 |            strFilename (string): file name to save data to
193 |            lstTbl (list of list): list of lists to hold data
194 |
195 |        Returns:
196 |            None
197 |
198 |        """
199 |        with open(strFilename, "wb") as objFile:
200 |            pickle.dump(lstTbl, objFile)
201 |
202 |    @staticmethod
203 |    def load_data(strFilename):
204 |        """Load data from file
205 |
206 |        Args:
207 |            strFilename (string): file name to load data from
208 |
209 |        Returns:
210 |            lstTbl (list of list): list of lists to hold data
211 |
212 |        """
213 |        with open(strFilename, "rb") as objFile:
214 |            lstTbl = pickle.load(objFile)
215 |
216 |    @staticmethod
217 |    def quit():
218 |        """Quit the program
219 |
220 |        Returns:
221 |            None
222 |
223 |        """
224 |        print("Thank you for using CDInventory. Goodbye!")
225 |
226 |    @staticmethod
227 |    def main():
228 |        """Main function
229 |
230 |        Returns:
231 |            None
232 |
233 |        """
234 |        # 1. when program starts, read in the currently saved inventory
235 |        lstTbl = FileProcessor.read_file(strFilename, lstTbl)
236 |
237 |        # 2. start main loop
238 |        while True:
239 |            # 2.1. Display menu to user and get choice
240 |            strChoice = IO.print_menu()
241 |
242 |            # 2.2. process menu selection
243 |            if strChoice == "1":
244 |                # 2.2.1. process add new CD
245 |                cd_id, title, artist = IO.get_user_input()
246 |                DataProcessor.user_add_cd_id, title, artist, lstTbl
247 |                IO.show_inventory(lstTbl)
248 |            elif strChoice == "2":
249 |                # 2.2.2. process delete CD
250 |                strIDdel = IO.del_input()
251 |                DataProcessor.user_del(strIDdel, lstTbl)
252 |                IO.show_inventory(lstTbl)
253 |            elif strChoice == "3":
254 |                # 2.2.3. process show inventory
255 |                IO.show_inventory(lstTbl)
256 |            elif strChoice == "4":
257 |                # 2.2.4. process save data
258 |                strFilename = input("Enter file name to save data to: ")
259 |                FileProcessor.save_data(strFilename, lstTbl)
260 |            elif strChoice == "5":
261 |                # 2.2.5. process load data
262 |                strFilename = input("Enter file name to load data from: ")
263 |                FileProcessor.load_data(strFilename)
264 |            elif strChoice == "6":
265 |                # 2.2.6. process quit
266 |                IO.quit()
267 |                break
268 |
269 |if __name__ == "__main__":
270 |    IO.main()
271 |
272 |"""GENERAL ERROR HANDLING"""
273 |def general_error():
274 |    """General error handling
275 |
276 |    Returns:
277 |        None
278 |
279 |    """
280 |    print("General Error")
281 |
282 |if __name__ == "__main__":
283 |    try:
284 |        IO.main()
285 |    except KeyboardInterrupt:
286 |        print("Program interrupted by user. Press (ENTER) to return to the menu.")
287 |    except EOFError:
288 |        print("Program ended by user. Press (ENTER) to return to the menu.")
289 |    except Exception as e:
290 |        general_error()
291 |
292 |if __name__ == "__main__":
293 |    IO.main()
294 |
295 |if __name__ == "__main__":
296 |    IO.main()
297 |
298 |if __name__ == "__main__":
299 |    IO.main()
300 |
301 |if __name__ == "__main__":
302 |    IO.main()
303 |
304 |if __name__ == "__main__":
305 |    IO.main()
306 |
307 |if __name__ == "__main__":
308 |    IO.main()
309 |
310 |if __name__ == "__main__":
311 |    IO.main()
312 |
313 |if __name__ == "__main__":
314 |    IO.main()
315 |
316 |if __name__ == "__main__":
317 |    IO.main()
318 |
319 |if __name__ == "__main__":
320 |    IO.main()
321 |
322 |if __name__ == "__main__":
323 |    IO.main()
324 |
325 |if __name__ == "__main__":
326 |    IO.main()
327 |
328 |if __name__ == "__main__":
329 |    IO.main()
330 |
331 |if __name__ == "__main__":
332 |    IO.main()
333 |
334 |if __name__ == "__main__":
335 |    IO.main()
336 |
337 |if __name__ == "__main__":
338 |    IO.main()
339 |
340 |if __name__ == "__main__":
341 |    IO.main()
342 |
343 |if __name__ == "__main__":
344 |    IO.main()
345 |
346 |if __name__ == "__main__":
347 |    IO.main()
348 |
349 |if __name__ == "__main__":
350 |    IO.main()
351 |
352 |if __name__ == "__main__":
353 |    IO.main()
354 |
355 |if __name__ == "__main__":
356 |    IO.main()
357 |
358 |if __name__ == "__main__":
359 |    IO.main()
360 |
361 |if __name__ == "__main__":
362 |    IO.main()
363 |
364 |if __name__ == "__main__":
365 |    IO.main()
366 |
367 |if __name__ == "__main__":
368 |    IO.main()
369 |
370 |if __name__ == "__main__":
371 |    IO.main()
372 |
373 |if __name__ == "__main__":
374 |    IO.main()
375 |
376 |if __name__ == "__main__":
377 |    IO.main()
378 |
379 |if __name__ == "__main__":
380 |    IO.main()
381 |
382 |if __name__ == "__main__":
383 |    IO.main()
384 |
385 |if __name__ == "__main__":
386 |    IO.main()
387 |
388 |if __name__ == "__main__":
389 |    IO.main()
390 |
391 |if __name__ == "__main__":
392 |    IO.main()
393 |
394 |if __name__ == "__main__":
395 |    IO.main()
396 |
397 |if __name__ == "__main__":
398 |    IO.main()
399 |
400 |if __name__ == "__main__":
401 |    IO.main()
402 |
403 |if __name__ == "__main__":
404 |    IO.main()
405 |
406 |if __name__ == "__main__":
407 |    IO.main()
408 |
409 |if __name__ == "__main__":
410 |    IO.main()
411 |
412 |if __name__ == "__main__":
413 |    IO.main()
414 |
415 |if __name__ == "__main__":
416 |    IO.main()
417 |
418 |if __name__ == "__main__":
419 |    IO.main()
420 |
421 |if __name__ == "__main__":
422 |    IO.main()
423 |
424 |if __name__ == "__main__":
425 |    IO.main()
426 |
427 |if __name__ == "__main__":
428 |    IO.main()
429 |
430 |if __name__ == "__main__":
431 |    IO.main()
432 |
433 |if __name__ == "__main__":
434 |    IO.main()
435 |
436 |if __name__ == "__main__":
437 |    IO.main()
438 |
439 |if __name__ == "__main__":
440 |    IO.main()
441 |
442 |if __name__ == "__main__":
443 |    IO.main()
444 |
445 |if __name__ == "__main__":
446 |    IO.main()
447 |
448 |if __name__ == "__main__":
449 |    IO.main()
450 |
451 |if __name__ == "__main__":
452 |    IO.main()
453 |
454 |if __name__ == "__main__":
455 |    IO.main()
456 |
457 |if __name__ == "__main__":
458 |    IO.main()
459 |
460 |if __name__ == "__main__":
461 |    IO.main()
462 |
463 |if __name__ == "__main__":
464 |    IO.main()
465 |
466 |if __name__ == "__main__":
467 |    IO.main()
468 |
469 |if __name__ == "__main__":
470 |    IO.main()
471 |
472 |if __name__ == "__main__":
473 |    IO.main()
474 |
475 |if __name__ == "__main__":
476 |    IO.main()
477 |
478 |if __name__ == "__main__":
479 |    IO.main()
480 |
481 |if __name__ == "__main__":
482 |    IO.main()
483 |
484 |if __name__ == "__main__":
485 |    IO.main()
486 |
487 |if __name__ == "__main__":
488 |    IO.main()
489 |
490 |if __name__ == "__main__":
491 |    IO.main()
492 |
493 |if __name__ == "__main__":
494 |    IO.main()
495 |
496 |if __name__ == "__main__":
497 |    IO.main()
498 |
499 |if __name__ == "__main__":
500 |    IO.main()
501 |
502 |if __name__ == "__main__":
503 |    IO.main()
504 |
505 |if __name__ == "__main__":
506 |    IO.main()
507 |
508 |if __name__ == "__main__":
509 |    IO.main()
510 |
511 |if __name__ == "__main__":
512 |    IO.main()
513 |
514 |if __name__ == "__main__":
515 |    IO.main()
516 |
517 |if __name__ == "__main__":
518 |    IO.main()
519 |
520 |if __name__ == "__main__":
521 |    IO.main()
522 |
523 |if __name__ == "__main__":
524 |    IO.main()
525 |
526 |if __name__ == "__main__":
527 |    IO.main()
528 |
529 |if __name__ == "__main__":
530 |    IO.main()
531 |
532 |if __name__ == "__main__":
533 |    IO.main()
534 |
535 |if __name__ == "__main__":
536 |    IO.main()
537 |
538 |if __name__ == "__main__":
539 |    IO.main()
540 |
541 |if __name__ == "__main__":
542 |    IO.main()
543 |
544 |if __name__ == "__main__":
545 |    IO.main()
546 |
547 |if __name__ == "__main__":
548 |    IO.main()
549 |
550 |if __name__ == "__main__":
551 |    IO.main()
552 |
553 |if __name__ == "__main__":
554 |    IO.main()
555 |
556 |if __name__ == "__main__":
557 |    IO.main()
558 |
559 |if __name__ == "__main__":
560 |    IO.main()
561 |
562 |if __name__ == "__main__":
563 |    IO.main()
564 |
565 |if __name__ == "__main__":
566 |    IO.main()
567 |
568 |if __name__ == "__main__":
569 |    IO.main()
570 |
571 |if __name__ == "__main__":
572 |    IO.main()
573 |
574 |if __name__ == "__main__":
575 |    IO.main()
576 |
577 |if __name__ == "__main__":
578 |    IO.main()
579 |
580 |if __name__ == "__main__":
581 |    IO.main()
582 |
583 |if __name__ == "__main__":
584 |    IO.main()
585 |
586 |if __name__ == "__main__":
587 |    IO.main()
588 |
589 |if __name__ == "__main__":
590 |    IO.main()
591 |
592 |if __name__ == "__main__":
593 |    IO.main()
594 |
595 |if __name__ == "__main__":
596 |    IO.main()
597 |
598 |if __name__ == "__main__":
599 |    IO.main()
600 |
601 |if __name__ == "__main__":
602 |    IO.main()
603 |
604 |if __name__ == "__main__":
605 |    IO.main()
606 |
607 |if __name__ == "__main__":
608 |    IO.main()
609 |
610 |if __name__ == "__main__":
611 |    IO.main()
612 |
613 |if __name__ == "__main__":
614 |    IO.main()
615 |
616 |if __name__ == "__main__":
617 |    IO.main()
618 |
619 |if __name__ == "__main__":
620 |    IO.main()
621 |
622 |if __name__ == "__main__":
623 |    IO.main()
624 |
625 |if __name__ == "__main__":
626 |    IO.main()
627 |
628 |if __name__ == "__main__":
629 |    IO.main()
630 |
631 |if __name__ == "__main__":
632 |    IO.main()
633 |
634 |if __name__ == "__main__":
635 |    IO.main()
636 |
637 |if __name__ == "__main__":
638 |    IO.main()
639 |
640 |if __name__ == "__main__":
641 |    IO.main()
642 |
643 |if __name__ == "__main__":
644 |    IO.main()
645 |
646 |if __name__ == "__main__":
647 |    IO.main()
648 |
649 |if __name__ == "__main__":
650 |    IO.main()
651 |
652 |if __name__ == "__main__":
653 |    IO.main()
654 |
655 |if __name__ == "__main__":
656 |    IO.main()
657 |
658 |if __name__ == "__main__":
659 |    IO.main()
660 |
661 |if __name__ == "__main__":
662 |    IO.main()
663 |
664 |if __name__ == "__main__":
665 |    IO.main()
666 |
667 |if __name__ == "__main__":
668 |    IO.main()
669 |
670 |if __name__ == "__main__":
671 |    IO.main()
672 |
673 |if __name__ == "__main__":
674 |    IO.main()
675 |
676 |if __name__ == "__main__":
677 |    IO.main()
678 |
679 |if __name__ == "__main__":
680 |    IO.main()
681 |
682 |if __name__ == "__main__":
683 |    IO.main()
684 |
685 |if __name__ == "__main__":
686 |    IO.main()
687 |
688 |if __name__ == "__main__":
689 |    IO.main()
690 |
691 |if __name__ == "__main__":
692 |    IO.main()
693 |
694 |if __name__ == "__main__":
695 |    IO.main()
696 |
697 |if __name__ == "__main__":
698 |    IO.main()
699 |
700 |if __name__ == "__main__":
701 |    IO.main()
702 |
703 |if __name__ == "__main__":
704 |    IO.main()
705 |
706 |if __name__ == "__main__":
707 |    IO.main()
708 |
709 |if __name__ == "__main__":
710 |    IO.main()
711 |
712 |if __name__ == "__main__":
713 |    IO.main()
714 |
715 |if __name__ == "__main__":
716 |    IO.main()
717 |
718 |if __name__ == "__main__":
719 |    IO.main()
720 |
721 |if __name__ == "__main__":
722 |    IO.main()
723 |
724 |if __name__ == "__main__":
725 |    IO.main()
726 |
727 |if __name__ == "__main__":
728 |    IO.main()
729 |
730 |if __name__ == "__main__":
731 |    IO.main()
732 |
733 |if __name__ == "__main__":
734 |    IO.main()
735 |
736 |if __name__ == "__main__":
737 |    IO.main()
738 |
739 |if __name__ == "__main__":
740 |    IO.main()
741 |
742 |if __name__ == "__main__":
743 |    IO.main()
744 |
745 |if __name__ == "__main__":
746 |    IO.main()
747 |
748 |if __name__ == "__main__":
749 |    IO.main()
750 |
751 |if __name__ == "__main__":
752 |    IO.main()
753 |
754 |if __name__ == "__main__":
755 |    IO.main()
756 |
757 |if __name__ == "__main__":
758 |    IO.main()
759 |
760 |if __name__ == "__main__":
761 |    IO.main()
762 |
763 |if __name__ == "__main__":
764 |    IO.main()
765 |
766 |if __name__ == "__main__":
767 |    IO.main()
768 |
769 |if __name__ == "__main__":
770 |    IO.main()
771 |
772 |if __name__ == "__main__":
773 |    IO.main()
774 |
775 |if __name__ == "__main__":
776 |    IO.main()
777 |
778 |if __name__ == "__main__":
779 |    IO.main()
780 |
781 |if __name__ == "__main__":
782 |    IO.main()
783 |
784 |if __name__ == "__main__":
785 |    IO.main()
786 |
787 |if __name__ == "__main__":
788 |    IO.main()
789 |
790 |if __name__ == "__main__":
791 |    IO.main()
792 |
793 |if __name__ == "__main__":
794 |    IO.main()
795 |
796 |if __name__ == "__main__":
797 |    IO.main()
798 |
799 |if __name__ == "__main__":
800 |    IO.main()
801 |
802 |if __name__ == "__main__":
803 |    IO.main()
804 |
805 |if __name__ == "__main__":
806 |    IO.main()
807 |
808 |if __name__ == "__main__":
809 |    IO.main()
810 |
811 |if __name__ == "__main__":
812 |    IO.main()
813 |
814 |if __name__ == "__main__":
815 |    IO.main()
816 |
817 |if __name__ == "__main__":
818 |    IO.main()
819 |
820 |if __name__ == "__main__":
821 |    IO.main()
822 |
823 |if __name__ == "__main__":
824 |    IO.main()
825 |
826 |if __name__ == "__main__":
827 |    IO.main()
828 |
829 |if __name__ == "__main__":
830 |    IO.main()
831 |
832 |if __name__ == "__main__":
833 |    IO.main()
834 |
835 |if __name__ == "__main__":
836 |    IO.main()
837 |
838 |if __name__ == "__main__":
839 |    IO.main()
840 |
841 |if __name__ == "__main__":
842 |    IO.main()
843 |
844 |if __name__ == "__main__":
845 |    IO.main()
846 |
847 |if __name__ == "__main__":
848 |    IO.main()
849 |
850 |if __name__ == "__main__":
851 |    IO.main()
852 |
853 |if __name__ == "__main__":
854 |    IO.main()
855 |
856 |if __name__ == "__main__":
857 |    IO.main()
858 |
859 |if __name__ == "__main__":
860 |    IO.main()
861 |
862 |if __name__ == "__main__":
863 |    IO.main()
864 |
865 |if __name__ == "__main__":
866 |    IO.main()
867 |
868 |if __name__ == "__main__":
869 |    IO.main()
870 |
871 |if __name__ == "__main__":
872 |    IO.main()
873 |
874 |if __name__ == "__main__":
875 |    IO.main()
876 |
877 |if __name__ == "__main__":
878 |    IO.main()
879 |
880 |if __name__ == "__main__":
881 |    IO.main()
882 |
883 |if __name__ == "__main__":
884 |    IO.main()
885 |
886 |if __name__ == "__main__":
887 |    IO.main()
888 |
889 |if __name__ == "__main__":
890 |    IO.main()
891 |
892 |if __name__ == "__main__":
893 |    IO.main()
894 |
895 |if __name__ == "__main__":
896 |    IO.main()
897 |
898 |if __name__ == "__main__":
899 |    IO.main()
900 |
901 |if __name__ == "__main__":
902 |    IO.main()
903 |
904 |if __name__ == "__main__":
905 |    IO.main()
906 |
907 |if __name__ == "__main__":
908 |    IO.main()
909 |
910 |if __name__ == "__main__":
911 |    IO.main()
912 |
913 |if __name__ == "__main__":
914 |    IO.main()
915 |
916 |if __name__ == "__main__":
917 |    IO.main()
918 |
919 |if __name__ == "__main__":
920 |    IO.main()
921 |
922 |if __name__ == "__main__":
923 |    IO.main()
924 |
925 |if __name__ == "__main__":
926 |    IO.main()
927 |
928 |if __name__ == "__main__":
929 |    IO.main()
930 |
931 |if __name__ == "__main__":
932 |    IO.main()
933 |
934 |if __name__ == "__main__":
935 |    IO.main()
936 |
937 |if __name__ == "__main__":
938 |    IO.main()
939 |
940 |if __name__ == "__main__":
941 |    IO.main()
942 |
943 |if __name__ == "__main__":
944 |    IO.main()
945 |
946 |if __name__ == "__main__":
947 |    IO.main()
948 |
949 |if __name__ == "__main__":
950 |    IO.main()
951 |
952 |if __name__ == "__main__":
953 |    IO.main()
954 |
955 |if __name__ == "__main__":
956 |    IO.main()
957 |
958 |if __name__ == "__main__":
959 |    IO.main()
960 |
961 |if __name__ == "__main__":
962 |    IO.main()
963 |
964 |if __name__ == "__main__":
965 |    IO.main()
966 |
967 |if __name__ == "__main__":
968 |    IO.main()
969 |
970 |if __name__ == "__main__":
971 |    IO.main()
972 |
973 |if __name__ == "__main__":
974 |    IO.main()
975 |
976 |if __name__ == "__main__":
977 |    IO.main()
978 |
979 |if __name__ == "__main__":
980 |    IO.main()
981 |
982 |if __name__ == "__main__":
983 |    IO.main()
984 |
985 |if __name__ == "__main__":
986 |    IO.main()
987 |
988 |if __name__ == "__main__":
989 |    IO.main()
990 |
991 |if __name__ == "__main__":
992 |    IO.main()
993 |
994 |if __name__ == "__main__":
995 |    IO.main()
996 |
997 |if __name__ == "__main__":
998 |    IO.main()
999 |
1000 |if __name__ == "__main__":
1001 |    IO.main()
1002 |
1003 |if __name__ == "__main__":
1004 |    IO.main()
1005 |
1006 |if __name__ == "__main__":
1007 |    IO.main()
1008 |
1009 |if __name__ == "__main__":
1010 |    IO.main()
1011 |
1012 |if __name__ == "__main__":
1013 |    IO.main()
1014 |
1015 |if __name__ == "__main__":
1016 |    IO.main()
1017 |
1018 |if __name__ == "__main__":
1019 |    IO.main()
1020 |
1021 |if __name__ == "__main__":
1022 |    IO.main()
1023 |
1024 |if __name__ == "__main__":
1025 |    IO.main()
1026 |
1027 |if __name__ == "__main__":
1028 |    IO.main()
1029 |
1030 |if __name__ == "__main__":
1031 |    IO.main()
1032 |
1033 |if __name__ == "__main__":
1034 |    IO.main()
1035 |
1036 |if __name__ == "__main__":
1037 |    IO.main()
1038 |
1039 |if __name__ == "__main__":
1040 |    IO.main()
1041 |
1042 |if __name__ == "__main__":
1043 |    IO.main()
1044 |
1045 |if __name__ == "__main__":
1046 |    IO.main()
1047 |
1048 |if __name__ == "__main__":
1049 |    IO.main()
1050 |
1051 |if __name__ == "__main__":
1052 |    IO.main()
1053 |
1054 |if __name__ == "__main__":
1055 |    IO.main()
1056 |
1057 |if __name__ == "__main__":
1058 |    IO.main()
1059 |
1060 |if __name__ == "__main__":
1061 |    IO.main()
1062 |
1063 |if __name__ == "__main__":
1064 |    IO.main()
1065 |
1066 |if __name__ == "__main__":
1067 |    IO.main()
1068 |
1069 |if __name__ == "__main__":
1070 |    IO.main()
1071 |
1072 |if __name__ == "__main__":
1073 |    IO.main()
1074 |
1075 |if __name__ == "__main__":
1076 |    IO.main()
1077 |
1078 |if __name__ == "__main__":
1079 |    IO.main()
1080 |
1081 |if __name__ == "__main__":
1082 |    IO.main()
1083 |
1084 |if __name__ == "__main__":
1085 |    IO.main()
1086 |
1087 |if __name__ == "__main__":
1088 |    IO.main()
1089 |
1090 |if __name__ == "__main__":
1091 |    IO.main()
1092 |
1093 |if __name__ == "__main__":
1094 |    IO.main()
1095 |
1096 |if __name__ == "__main__":
1097 |    IO.main()
1098 |
1099 |if __name__ == "__main__":
1100 |    IO.main()
1101 |
1102 |if __name__ == "__main__":
1103 |    IO.main()
1104 |
1105 |if __name__ == "__main__":
1106 |    IO.main()
1107 |
1108 |if __name__ == "__main__":
1109 |    IO.main()
1110 |
1111 |if __name__ == "__main__":
1112 |    IO.main()
1113 |
1114 |if __name__ == "__main__":
1115 |    IO.main()
1116 |
1117 |if __name__ == "__main__":
1118 |    IO.main()
1119 |
1120 |if __name__ == "__main__":
1121 |    IO.main()
1122 |
1123 |if __name__ == "__main__":
1124 |    IO.main()
1125 |
1126 |if __name__ == "__main__":
1127 |    IO.main()
1128 |
1129 |if __name__ == "__main__":
1130 |    IO.main()
1131 |
1132 |if __name__ == "__main__":
1133 |    IO.main()
1134 |
1135 |if __name__ == "__main__":
1136 |    IO.main()
1137 |
1138 |if __name__ == "__main__":
1139 |    IO.main()
1140 |
1141 |if __name__ == "__main__":
1142 |    IO.main()
1143 |
1144 |if __name__ == "__main__":
1145 |    IO.main()
1146 |
1147 |if __name__ == "__main__":
1148 |    IO.main()
1149 |
1150 |if __name__ == "__main__":
1151 |    IO.main()
1152 |
1153 |if __name__ == "__main__":
1154 |    IO.main()
1155 |
1156 |if __name__ == "__main__":
1157 |    IO.main()
1158 |
1159 |if __name__ == "__main__":
1160 |    IO.main()
1161 |
1162 |if __name__ == "__main__":
1163 |    IO.main()
1164 |
1165 |if __name__ == "__main__":
1166 |    IO.main()
1167 |
1168 |if __name__ == "__main__":
1169 |    IO.main()
1170 |
1171 |if __name__ == "__main__":
1172 |    IO.main()
1173 |
1174 |if __name__ == "__main__":
1175 |    IO.main()
1176 |
1177 |if __name__ == "__main__":
1178 |    IO.main()
1179 |
1180 |if __name__ == "__main__":
1181 |    IO.main()
1182 |
1183 |if __name__ == "__main__":
1184 |    IO.main()
1185 |
1186 |if __name__ == "__main__":
1187 |    IO.main()
1188 |
1189 |if __name__ == "__main__":
1190 |    IO.main()
1191 |
1192 |if __name__ == "__main__":
1193 |    IO.main()
1194 |
1195 |if __name__ == "__main__":
1196 |    IO.main()
1197 |
1198 |if __name__ == "__main__":
1199 |    IO.main()
1200 |
1201 |if __name__ == "__main__":
1202 |    IO.main()
1203 |
1204 |if __name__ == "__main__":
1205 |    IO.main()
1206 |
1207 |if __name__ == "__main__":
1208 |    IO.main()
1209 |
1210 |if __name__ == "__main__":
1211 |    IO.main()
1212 |
1213 |if __name__ == "__main__":
1214 |    IO.main()
1215 |
1216 |if __name__ == "__main__":
1217 |    IO.main()
1218 |
1219 |if __name__ == "__main__":
1220 |    IO.main()
1221 |
1222 |if __name__ == "__main__":
1223 |    IO.main()
1224 |
1225 |if __name__ == "__main__":
1226 |    IO.main()
1227 |
1228 |if __name__ == "__main__":
1229 |    IO.main()
1230 |
1231 |if __name__ == "__main__":
1232 |    IO.main()
1233 |
1234 |if __name__ == "__main__":
1235 |    IO.main()
1236 |
1237 |if __name__ == "__main__":
1238 |    IO.main()
1239 |
1240 |if __name__ == "__main__":
1241 |    IO.main
```

Example Run from Spyder

```
In [1]: runfile('C:/Python/Assignment07/CDInventory_07.py', wdir='C:/Python/Assignment07')
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]: 1

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.

type 'yes' to continue and reload from file. otherwise reload will be canceled: yes
reloading...
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====
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

Enter ID: song
Not an integer
Build in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'song'
Inappropriate argument value (of correct type).
```

```
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====

Enter ID: 5

What is the CD's title? Crashing

What is the Artist's name? Illenium
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
5       Crashing (by:Illenium)
=====
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]: d

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
5       Crashing (by:Illenium)
=====

Which ID would you like to delete? delete
Not an integer
Build in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'delete'
Inappropriate argument value (of correct type).

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
5       Crashing (by:Illenium)
=====

Which ID would you like to delete? 5
The CD was removed
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====
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]: x
```

Example Run from Terminal

```
Anaconda Powershell Prompt (Anaconda3)

(base) PS C:\Users\CASE> cd C:\_Python\
(base) PS C:\_Python> cd .\Assignment07\
(base) PS C:\_Python\Assignment07> python .\cdInventory_07.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]: 1

WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.
Type 'yes' to continue and reload from file, otherwise reload will be canceled: yes
reloading...

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====
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

Enter ID: id
Not an integer:
Build in error info:
<class 'ValueError':
  invalid literal for int() with base 10: 'id'
Inappropriate argument value (of correct type).

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====
Enter ID: 5
what is the CD's title? songs
what is the Artist's name? artists
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
5       songs (by:artists)
=====
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]: d

===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
5       songs (by:artists)
=====
Which ID would you like to delete? 5
The CD was removed
===== The Current Inventory: =====
ID      CD Title (by: Artist)
1       The Search (by:AJR)
2       Paralyzed (by:NF)
3       Everything (by:Michael Buble)
4       Rewind (by:Louis Futon)
=====
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]: x

(base) PS C:\_Python\Assignment07>
```