<Ziqi Yan>

<2022/3/20>

<IT FDN 110>

<Assignment\_08>

CD Inventory

# Introduction

The Assignment 08 is about implementation of python classes. Based on assignment\_06, I learned making properties, methods, and constructor in a python class. And the concept of static methods.

# Topic\_1 Properties and Constructor

I learned writing properties started from writing default values. After I learned implementing \_\_init\_\_ function, I realized there is no need to define properties. So I integrated them inside the init function:

Text

Description automatically generated

Figure 1 CD class and the init constructor

# Topic\_2 Static methods: FileIO

For properties in class which need to be instantiate like CD’s cd\_id, we must create an object of it before using its properties. But in some scenarios, the methods/properties are not related to any instance of the class. Even the class is an abstract concept which is hard to instantiate. For those methods we have decorator: @staticmethods

Text

Description automatically generated

Figure 2 Static method load\_inventory in class FileIO

Text

Description automatically generated

Figure 3 static method show\_menu in class IO

# Topic\_3 docstring

In my opinion, docstring is a special type of comments that the IDE or editor will parse it as the description of the methods( or classes).

Text

Description automatically generated

Figure 4 docstring and its look in vscode

# Summery

In this assignment. I learned how to define and implement classes and documenting them with docstring. And I also got how to use @staticmethod decorator.

# Appendix

CD

class CD:

"""Stores data about a CD:

properties:

cd\_id: (int) with CD ID

cd\_title: (string) with the title of the CD

cd\_artist: (string) with the artist of the CD

methods:

"""

def \_\_init\_\_(self, id, title, artist):

self.cd\_id = id

self.cd\_title = title

self.cd\_artist = artist

FileIO

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 load\_inventory(file\_name):

"""Function to manage data ingestion from file to a list of dictionaries

Reads the data from file identified by file\_name into a 2D table

(list of dicts) table one line in the file represents one dictionary row in table.

Args:

file\_name (string): name of file used to read the data from

table (list of dict): 2D data structure (list of dicts) that holds the data during runtime

Returns:

None.

"""

tables = [] # this clears existing data and allows to load data from file

objFile = open(file\_name, 'r')

for line in objFile:

data = line.strip().split(',')

dicRow = CD(int(data[0]), data[1], data[2])

tables.append(dicRow)

objFile.close()

return tables

@staticmethod

def save\_inventory(file\_name, lst\_Inventory):

objFile = open(file\_name, 'w')

for cd in lst\_Inventory:

lstValues = [cd.cd\_id, cd.cd\_title, cd.cd\_artist]

lstValues[0] = str(lstValues[0])

objFile.write(','.join(lstValues) + '\n')

objFile.close()

IO

class IO:

"""Processes data to and from file:

properties:

methods:

show\_menu(): -> None

get\_choice(): -> String

dis\_data(lst\_Inventory): -> None

add\_cd(lst\_Inventory): -> None

"""

@staticmethod

def show\_menu():

"""Displays a menu of choices to the user

Args:

None.

Returns:

None.

"""

print('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')

print('[s] Save Inventory to file\n[x] exit\n')

@staticmethod

def get\_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, i, d, s or x

"""

choice = ' '

while choice not in ['l', 'a', 'i', 'd', 's', 'x']:

choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()

print() # Add extra space for layout

return choice

@staticmethod

def dis\_data(lst\_Inventory):

"""Displays current inventory table

Args:

table (list of cd object): list of cd objects that holds the data during runtime.

Returns:

None.

"""

print('======= The Current Inventory: =======')

print('ID\tCD Title (by: Artist)\n')

for cd in lst\_Inventory:

print('{}\t{} (by:{})'.format(\*[cd.cd\_id, cd.cd\_title, cd.cd\_artist]))

print('======================================')

# TODO add code to get CD data from user

@staticmethod

def add\_cd(lst\_Inventory):

strID = input('Enter ID: ').strip()

strTitle = input('What is the CD\'s title? ').strip()

stArtist = input('What is the Artist\'s name? ').strip()

lst\_Inventory.append(CD(int(strID), strTitle, stArtist))