from collections import namedtuple

fields = namedtuple('fields', ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model\_year', 'origin', 'car\_name'])

import csv

import os

import requests

import logging

import argparse

from operator import attrgetter

logger=logging.getLogger()

logger.setLevel(logging.DEBUG)

fileHandle = logging.FileHandler('autompg2.log', 'w')

fileHandle.setLevel(logging.DEBUG)

logger.addHandler(fileHandle)

streamHandle= logging.StreamHandler()

streamHandle.setLevel(logging.INFO)

logger.addHandler(streamHandle)

class AutoMPG:

"""Class that represents the attrributes available for each record in a data set"""

def \_\_init\_\_(self, make, model, year, mpg):

self.make=str(make)

self.model=str(model)

self.year=1900+int(year)

self.mpg=float(mpg)

def \_\_str\_\_(self):

#string representation of the record

return f'AutoMPG({self.make},{self.model}, {self.year}, {self.mpg})'

def \_\_repr\_\_(self):

#string representation of the record

return self.\_\_str\_\_()

def \_\_eq\_\_(self, other):

#compare the make model year and mpg of two records (equal)

#ensure types are the same

if type(self) == type(other):

return self.make == other.make and self.model == other.model and self.year==other.year and self.mpg==other.mpg

else:

return NotImplemented

def \_\_lt\_\_(self, other):

#compare the make model year and mpg of two records (less than)

#ensure types are the same

if type(self) == type(other):

#compare the first attribute: make

if self.make != other.make:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the second attribute: model

if self.model != other.model:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the third attribute: year

if self.year != other.year:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

else:

#compare the fourth attribute: mpg

if self.mpg != other.mpg:

return (self.make, self.model, self.year, self.mpg) < (other.make, other.model, other.year, other.mpg)

#if all 4 attributes are the same, comparision not implemented

else:

return NotImplemented

else:

return NotImplemented

def \_\_hash\_\_(self):

#returns the hash of all the attributes

return hash((self.make, self.model, self.year, self.mpg))

class AutoMPGData:

"""Class that represents the entire AutoMPG data set. Data is a list of AutoMPG objects"""

def \_\_init\_\_(self):

self.data=self.\_load\_data()

def \_\_iter\_\_(self):

#allow iteration through the list

self.\_iter = 0

return self

def \_\_next\_\_(self):

#next is each record in the list

if self.\_iter == len(self.data):

raise StopIteration

ret = self.data[self.\_iter]

self.\_iter +=1

return ret

def \_load\_data(self):

if os.path.exists("auto-mpg.data.txt")== False:

logging.debug('checking auto-mpg.data.txt')

self.\_get\_data()

#check if the file already exists

if os.path.exists("auto-mpg.clean.txt")== False:

logging.debug('checking auto-mpg.clean.txt')

#if file DNE, send to clean data

self.\_clean\_data()

else:

logging.debug('auto-mpg.clean.txt exits')

#used cleaned data to create the data list

data=[]

#take the data from the cleaned CSV

with open('auto-mpg.clean.txt', 'r', newline='') as csvfile:

dataReader = csv.reader(csvfile)

logging.debug('Parsing auto-mpg.clean.txt into AutoMPG objects')

for row in dataReader:

#put record into a tuple and unpack the tuple

field = fields(row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8])

mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name = field

#add the record to the data list

car=Record(mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name)

data+=[car.pass\_into\_autoMPG()]

return data

def \_clean\_data(self):

#create a new data file with the cleaned data

#remove the tab

data\_file = csv.register\_dialect("space",delimiter='\t', skipinitialspace = True)

with open('auto-mpg.clean.txt', 'w', newline='') as outputfile:

cleanwriter = csv.writer(outputfile, quoting=csv.QUOTE\_NONE)

#take the data from the original AutoMPG CSV file

with open('auto-mpg.data.txt', 'r', newline='') as csvfile:

dataReader = csv.reader(csvfile, "space")

for row in dataReader:

#split all attributes other than car\_name

clean=row[0].split() + [row[1]]

cleanwriter.writerow(clean)

self.\_load\_data()

def \_get\_data(self):

logging.debug('getting auto-mpg.data.txt')

url='https://autompgdata.azurewebsites.net/api/assignment7data'

r = requests.get(url, allow\_redirects = True)

code=r.status\_code

logging.debug('Responce code: %s', r.status\_code)

with open('auto-mpg.data.txt', 'wb') as output:

output.write(r.content)

self.\_load\_data()

return code

def sort\_by\_default(self):

logging.debug('Sorting AutoMPG objects by default')

self.data.sort()

return self.data

def sort\_by\_year(self):

logging.debug('Sorting AutoMPG objects by year')

self.data.sort(key= attrgetter('year', 'make', 'model', 'mpg'))

return self.data

def sort\_by\_mpg(self):

logging.debug('Sorting AutoMPG objects by mpg')

self.data.sort(key= attrgetter('mpg', 'make', 'model', 'year'))

return self.data

class Record:

"""Class that passes the appropriate attributes into the AutoMPG class"""

def \_\_init\_\_(self, mpg, cylinders, displacement, horsepower, weight, acceleration, model\_year, origin, car\_name):

self.mpg=mpg

self.cylinders=cylinders

self.displacement=displacement

self.horsepower=horsepower

self.weight=weight

self.acceleration=acceleration

self.model\_year=model\_year

self.origin=origin

self.car\_name=car\_name

def pass\_into\_autoMPG(self):

#split car\_name into make and model

split\_names=self.car\_name.split(" ", 1)

make=split\_names[0]

if split\_names[-1]== split\_names[0]:

model=''

else:

model=split\_names[-1]

#pass attributes into AutoMPG

auto=AutoMPG(make,model,self.model\_year , self.mpg)

return auto

def main():

parser =argparse.ArgumentParser(description='Parse the data from specificed access log')

parser.add\_argument ('command', metavar='command' , type=str, help='command to execute', choices=['print'])

parser.add\_argument('-s', '--sort',metavar= '<sort order>', type=str, choices=[None, 'mpg', 'year'])

arguments= parser.parse\_args()

car=AutoMPGData()

if arguments.sort== None:

output=car.sort\_by\_default()

if arguments.sort == 'mpg':

output=car.sort\_by\_mpg()

if arguments.sort == 'year':

output=car.sort\_by\_year()

for a in output:

logging.info(a)

if \_\_name\_\_=='\_\_main\_\_':

main()