Kevin Leary

Homework 3

**Problem 1**

import csv

#function to read imdb-top-rated

def read\_top\_rated(top\_rated\_file, top\_rated\_dict):

"""Read the imdb\_top\_rated.csv to create a dictionary

with the tuple key (title, year):imdb\_rating """

data\_reader = csv.reader(top\_rated\_file)

for row in data\_reader:

# ignore header rows: elements begin with a number since sorted by rank

#row increments in for and col is the index i.e. [0]

if row[0].isdigit():

#rank = row[0]

title = row[1]

year = row[2]

imdb\_rating = row[3]

top\_rated\_dict[(title, year)] = imdb\_rating

#print(top\_rated\_dict) #-- looks good!

#function to read imdb-top-grossing

def read\_top\_grossing(top\_grossing\_file, top\_grossing\_dict):

"""Read the imdb\_top\_grossing.csv to create a dictionary

with the tuple key (title, year):box\_office """

data\_reader = csv.reader(top\_grossing\_file)

for row in data\_reader:

# ignore header rows: elements begin with a number

if row[0].isdigit():

title = row[1]

year = row[2]

box\_office = row[3]

#top\_grossing\_dict = {(title, year):box\_office}

#print(top\_grossing\_dict)

top\_grossing\_dict[(title, year)] = box\_office

#function to read imdb-top-casts

def read\_top\_casts(top\_casts\_file, top\_casts\_dict):

"""Read the imdb\_top\_casts.csv to create a dictionary

with the tuple key (title, year):

--- this one is a little less straight foward"""

data\_reader = csv.reader(top\_casts\_file)

for row in data\_reader:

# ignore header rows: elements begin with a number

#if row[0].isdigit(): #---not needed because no headers for this csv

title = row[0]

year = row[1]

director = row[2]

cast = row[3:8]

#top\_casts\_dict = {(title, year):(director, cast)}

top\_casts\_dict[(title, year)] = (director, cast)

#top\_casts\_dict = {(title, year):director}

#print(top\_casts\_dict)

#top\_rated\_dict = {('movie', 'year'):[rating]}

#Read the files

top\_rated\_file = open(r'E:\A FAU\COP4045 - Python\HW\HW3\imdb-top-rated.csv')

top\_grossing\_file = open(r'E:\A FAU\COP4045 - Python\HW\HW3\imdb-top-grossing.csv')

top\_casts\_file = open(r'E:\A FAU\COP4045 - Python\HW\HW3\imdb-top-casts.csv', encoding='utf8') #pain in the ass

#create dictionaries

top\_rated\_dict = {}

read\_top\_rated(top\_rated\_file, top\_rated\_dict)

top\_grossing\_dict = {}

read\_top\_grossing(top\_grossing\_file, top\_grossing\_dict)

top\_casts\_dict = {}

read\_top\_casts(top\_casts\_file, top\_casts\_dict)

#1a) print the directors with the most movies in the top rated list

dir\_count = {}

i = 0

#query values and put them in a new dict with a count

for key, value in top\_casts\_dict.items():

if key in top\_rated\_dict:

valnew = top\_casts\_dict[key][0]

dir\_count[valnew] = dir\_count.get(valnew, 0) + 1

#reverse sort the values

dir\_sorted = sorted(dir\_count.items(), key=lambda kv:kv[1], reverse=True)

#print values

print("Directors with most movies in the top rated\n")

while(i < 5):

print(dir\_sorted[i][0])

i+= 1

#1b) directors with most movies in the top grossing

dir\_gross\_count = {}

i = 0

#query values and put them in a new dict with a count

for key, value in top\_casts\_dict.items():

if key in top\_grossing\_dict:

valnew = top\_casts\_dict[key][0]

dir\_gross\_count[valnew] = dir\_gross\_count.get(valnew, 0) + 1

#reverse sort the values

dir\_gross\_sorted = sorted(dir\_gross\_count.items(), key=lambda kv:kv[1], reverse=True)

#print values

print("\n\nDirectors with most highest grossing movies\n")

while(i < 5):

print(dir\_gross\_sorted[i][0])

i+= 1

#1c) displays a ranking of the actors with the most movie credits in top rated

actor\_count = {}

i=0

for key, value in top\_casts\_dict.items():

if key in top\_rated\_dict:

for name in top\_casts\_dict[key][1]:

actor\_count[name] = actor\_count.get(name, 0) + 1

actor\_count\_sorted = sorted(actor\_count.items(), key=lambda kv:kv[1], reverse=True)

#print values

print("\n\nMost credited actors in top rated\n")

while(i < 5):

print(actor\_count\_sorted[i][0])

i+= 1

#1d) Displays a ranking (descending) with the actors who brought in the most box office money,

# based onthe top grossing movie list. For a movie with gross ticket sales amount s,

# the 5 actors on the cast list will split amount s in the following way:

actor\_gross\_count = {}

i=0

for key, value in top\_casts\_dict.items():

if key in top\_grossing\_dict:

for name in top\_casts\_dict[key][1]:

#for the splits

#would have to convert the money

actor\_gross\_count[name] = top\_grossing\_dict

actor\_count\_sorted = sorted(actor\_count.items(), key=lambda kv:kv[1], reverse=True)

#print values

print("\n\nMost credited actors in top rated\n")

while(i < 5):

print(actor\_count\_sorted[i][0])

i+= 1

**Problem 2**

class poly:

#constructor

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

#initializing the variables -- cast to float

self.coefs = [float(coefs) for coefs in coefs]

self.degree=len(coefs)-1

self.rep = self.\_\_str\_\_()

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

#for string conversion of the polynomial

if len(self.coefs)==0:

return str(0)

polynomial=''

if self.coefs[0]!=0:

if self.coefs[0]<0:

polynomial += str(self.coefs[0])

else:

polynomial += '+' + str(self.coefs[0])

if self.coefs[1]!=0:

if self.coefs[1]<0:

polynomial += str(self.coefs[1])+'X'

else:

polynomial += '+' + str(self.coefs[1])+ 'X'

index = 2

while index < len(self.coefs):

if self.coefs[index] != 0:

if self.coefs[index] < 0:

polynomial += str(self.coefs[index]) + 'X^' + str(index)

else:

polynomial += '+' + str(self.coefs[index]) + 'X^' + str(index)

index+=1

return polynomial

#for printing

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

return self.rep

#takes the coeffcient parameters

def \_\_getitem\_\_(self, k):

if k < len(self.coefs):

return self.coefs[k]

#does addition with another poly

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

polysum=[]

for i in range(len(self.coefs)):

polysum.append(self.coefs[i] + other.coefs[i])

return poly(polysum)

#multiplication with poly for float

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

num = self.degree + other.degree

product = [0]\*(num+1)

for i in range(0, self.degree + 1):

for j in range(0, other.degree + 1):

product[i+j] += self.coefs[i] \* other.coefs[j]

return poly(product)

#multiplication for poly ---int

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

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

if type(other) == int or type(other) == float:

for x in self.coefs:

return other\*x

else:

raise NotImplementedError

#test two polys for equality

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

if len(self.coefs)!=len(other.coefs):

print('False')

return False

elif len(self.coefs)==len(other.coefs):

for i in range(len(self.coefs)):

if self.coefs[i] != other.coefs[i]:

print('False')

return False

print('True')

return True

#test two polys for equality

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

if len(self.coefs)!=len(other.coefs):

print('True')

return True

elif len(self.coefs)==len(other.coefs):

for i in range(len(self.coefs)):

if self.coefs[i] != other.coefs[i]:

print('True')

return True

print('False')

return False

#evaluation for current poly

def eval(self, x):

eval=0

if type(x) == list:

print('in eval')

eval = [self.\_\_init\_\_(i) for i in x]

return eval

elif type(x)==float or type(x)==int:

for i in range(len(self.coefs)):

eval += self.coefs[i] \* pow(x, i)#

return eval

def test\_poly():

p1 = poly([1, -2, 1]) # poly of grade 2: p1(X)=1-2X+X2

p2 = poly((0, 1)) # create poly of grade 1 with a tuple: p2(X)=X, (a0==0)

print(p1) # print calls \_\_str\_\_ and prints 1.0-2.0X+X^2

p1 # python calls \_\_repr\_\_ and displays 1.0-2.0X+X^2

p1 == p2 # returns False

p1 == poly((1, -2, 1)) # return True

p1 != p2 # returns True

p3 = p1 + p2 # sum, \_\_add\_\_

print(p3) # prints 1.0-X+X^2.0 (use default number of decimals)

p1[1] # indexing the coefficients: returns -2 (a1 for p1)

p4 = p2 \* p1 # product with another Poly: p4 becomes X-2X^2+X^3

p5 = p1 \* 2 # product with int or float: p5 becomes 2-4X+2X^2

p6 = 3 \* p1 # product with int or float: p6 becomes 3-6X+3X^2 (\_\_rmul\_\_)

print( p1.eval(2) ) # evaluate p1 at point 2: prints 1.0

print( p1.eval([0,-1,1]) # evaluate p1 for a list of points: prints [1,4,0]

test\_poly() #wtf

**Problem 3**

class Employee():

#constructor to initialize the object

def \_\_init\_\_(self,name,salary,phone):

self.\_\_name = name

self.\_\_base\_salary = salary

self.\_\_phone\_number = phone

#getter for name

def get\_name(self):

return self.\_\_name

#getter for number

def get\_number(self):

return self.\_\_phone\_number

#mutator function for salary

def mutator(self,end\_salary):

self.\_\_base\_salary = end\_salary

#calculates the total

def salary\_total(self):

return self.\_\_base\_salary

#to make the infor a string

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

return "Employee({}), {}, {}".format(self.\_\_name,self.\_\_phone\_number,self.salary\_total())

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

return self.\_\_str\_\_

class Manager(Employee):

#manger class works off of the employee class

#constuctor for this class object

def \_\_init\_\_(self, name, salary, phone,bonus):

super().\_\_init\_\_(name,salary,phone)

self.\_\_bonus=bonus

def salary\_total(self):

return super().salary\_total()+self.\_\_bonus

class Engineer(Employee):

#engineer class works off the employee class

def \_\_init\_\_(self,name,salary,phone):

super().\_\_init\_\_(name,salary,phone)

class CEO(Manager):

#ceo object funtions like a manager

def \_\_init\_\_(self,name,base\_salary,phone,bonus,stock):

super().\_\_init\_\_(name,base\_salary,phone,bonus)

self.\_\_stock\_options=stock

def salary\_total(self):

return super().salary\_total()+self.\_\_stock\_options

#function for printing the objct

def print\_staff(staff):

for i in staff:

print(i)

#initalize objects then print them to terminal

employee = Employee("Jayson Tatum", 100000, "1-800-932-0987")

engineer = Engineer("Marcus Smart", 12000000, "973-219-0441")

manager = Manager("Bob Roland", 40000, "286-813-9712", 5000)

ceo = CEO("Kevin Leary", 12000000, "000-000-0001", 55000, 78000)

result = [employee, engineer, manager, ceo]

print\_staff(result)