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)
```