**Homework 4**

Problem 1. Top Movies and Actors

Top-Down Design:

1. Display rank of top 5 directors with movies in top-rated.
   1. Get top rated list from tr-file
   2. Get directors from tc-file
   3. Keep count of director names
2. Display rank of top 5 directors with movies in top-gross.
   1. Get top gross list from tr-file
   2. Get directors from tc-file
   3. For movie in top gross
3. Display rank of top 5 actors with movies in top-rated.
   1. Make dictionary holding actor names with a list of movies as key.
4. Display rank of top 5 actors who brought money.

Shared Functionality:

1. Reading a file with a header and type-casting a line with int, str, int, float.
2. Making a dictionary with key = (mv, yr).
3. Counting occurrences in dictionary.
4. Using a movie as key to access a value in another dictionary.

# -\*- coding: utf-8 -\*-

'''Top Movies and Cast'''

import csv

def print\_directors(title, value, lst):

print('-'\*80)

d = '-'\*20

c = '-'\*5

print('{:<20s} | {:<5s}'.format(title, value))

print('{:<20s} | {:<5s}'.format(d, c))

for i, row in enumerate(lst):

print('{:<20s} | {:<5d}'.format(row[1], row[0]))

if i == 5:

break

print('\n\n')

def print\_actors(title, value, lst):

print('-'\*80)

d = '-'\*20

c = '-'\*20

print('{:<20s} | {:<5s}'.format(title, value))

if value == 'Count':

print('{:<20s} | {:<5s}'.format(d, c))

else:

print('{:<20s} | {:<20s}'.format(d, c))

for i, row in enumerate(lst):

if value == 'Count':

print('{:<20s} | {:<5d}'.format(row[1], row[0]))

else:

print('{:<20s} | {:<20.2f}'.format(row[1], row[0]))

if i == 5:

break

print('\n\n')

def prdict(dct):

print('\n'\*2)

for i, (k, v) in enumerate(dct.items()):

print(k,' : ', v)

if i == 9:

print('\n'\*2)

break

def get\_lists():

'''Returns 3 lists in this order: top-rated, top-gross, top-casts'''

# File Parsing: read\_ranks() handles both top-gross and top-rated

def read\_casts(afile):

data = csv.reader(afile)

return [tuple(row) for row in data]

def read\_ranks(afile):

'''types: int, str, int, float. Header is skipped.'''

def parse\_row(p):

'''p = position'''

return int(p[0]), p[1], int(p[2]), float(p[3])

data = csv.reader(afile)

next(data)

return [tuple(parse\_row(row)) for row in data]

file\_top\_rated = 'imdb-top-rated.csv'

file\_top\_gross = 'imdb-top-grossing.csv'

file\_top\_casts = 'imdb-top-casts.csv'

with open(file\_top\_rated, 'r', encoding='utf-8') as fobj:

rank\_ls = read\_ranks(fobj)

with open(file\_top\_gross, 'r', encoding='utf-8') as fobj:

gross\_ls = read\_ranks(fobj)

with open(file\_top\_casts, 'r', encoding='utf-8') as fobj:

cast\_ls = read\_casts(fobj)

return rank\_ls, gross\_ls, cast\_ls

def get\_dicts(ls1, ls2, ls3):

def get\_rank\_dict(lst):

return {(p[1], p[2]): (p[0], p[3]) for p in lst}

def get\_cast\_dict(lst):

return {(p[0], int(p[1])): (p[2:]) for p in lst}

def get\_actors\_dict(dct):

'''Returns a dictionary: key='actorname' value=<list of tuples>'''

ac\_mvs = {}

for movie, cast in dct.items():

for actor in cast[1:]:

if actor in ac\_mvs:

ac\_mvs[actor].append(movie)

else:

ac\_mvs[actor] = [movie]

return ac\_mvs

rd = get\_rank\_dict(ls1)

gd = get\_rank\_dict(ls2)

cd = get\_cast\_dict(ls3)

am = get\_actors\_dict(cd)

return rd, gd, cd, am

def count\_listings(cast\_dct, rank\_dct):

count = {}

rset = set(rank\_dct.keys())

for key in rset:

name = cast\_dct[key][0]

count[name] = count.get(name, 0) + 1

return sorted([(num, name) for name, num in count.items()], reverse=True)

def count\_roles(actors\_dct, rank\_dct):

'''Returns a list of (actors, #roles in top rated)'''

count = {}

rset = set(rank\_dct.keys())

for (act, mvs) in actors\_dct.items():

for mv in mvs:

if mv in rset:

count[act] = count.get(act, 0) + 1

return sorted([(num, name) for name, num in count.items()], reverse=True)

def top\_earners(gross\_dct, cast\_dct):

a = {}

def amt(n, i, s):

return ((2\*\*(n-i))\*s) / 31

for movie in gross\_dct.keys():

gross = gross\_dct[movie][1]

actors = cast\_dct[movie][1:]

n = len(actors)

for i, actor in enumerate(actors):

a[actor] = a.get(actor, 0) + amt(n, i+1, gross)

return sorted([(v, k) for k, v in a.items()], reverse=True)

def main():

rank\_ls, gross\_ls, cast\_ls = get\_lists()

rank\_dct, gross\_dct, cast\_dct, act\_roles\_dct = get\_dicts(rank\_ls, gross\_ls, cast\_ls)

dir\_in\_top\_rated = count\_listings(cast\_dct, rank\_dct)

dir\_in\_top\_gross = count\_listings(cast\_dct, gross\_dct)

act\_in\_top\_rated = count\_roles(act\_roles\_dct, rank\_dct)

act\_in\_top\_gross = top\_earners(gross\_dct, cast\_dct)

print('\n\n')

print('Directors with most movies in top-rated list')

print\_directors('Directors', 'Count', dir\_in\_top\_rated)

print('\n\n')

print('Directors with most movies in top-grossing list')

print\_directors('Directors', 'Count', dir\_in\_top\_gross)

print('\n\n')

print('Actors with most movies in top-rated list')

print\_actors('Actors', 'Count', act\_in\_top\_rated)

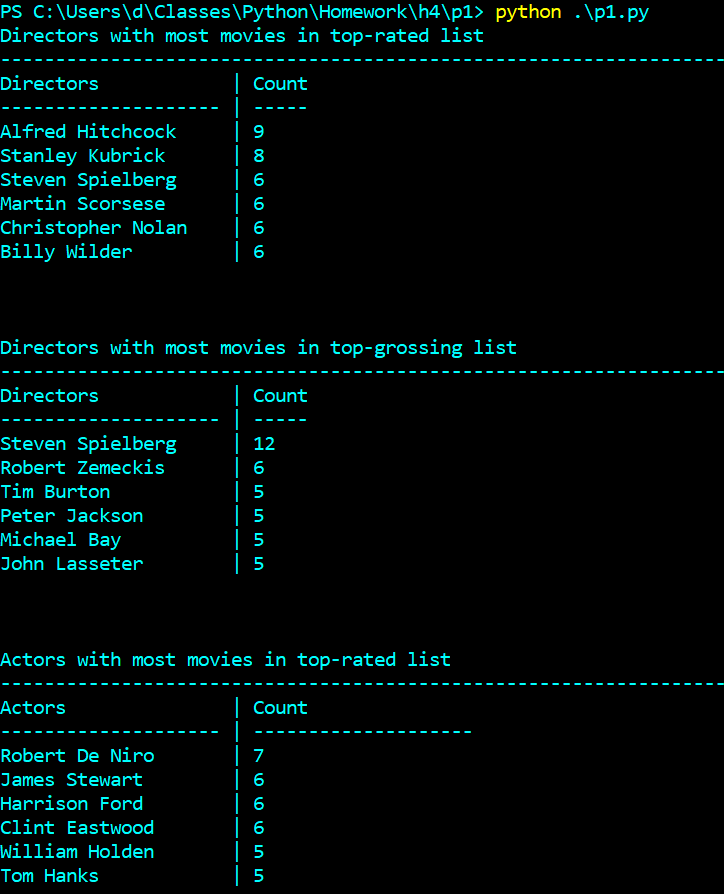
print('\n\n')

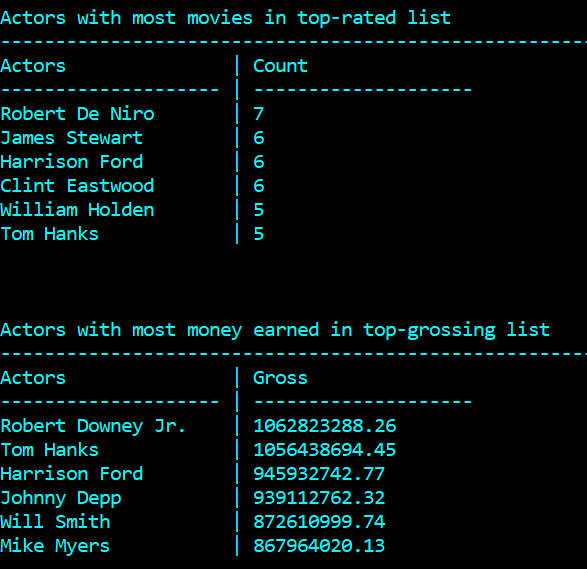
print('Actors with most money earned in top-grossing list')

print\_actors('Actors', 'Gross', act\_in\_top\_gross)

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

main()





Problem 2. Polynomial Class

# -\*- coding: utf-8 -\*-

class Poly(object):

'''Takes a list of coefficients to make a working polynomial type'''

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

'''Assumes input is an integer or float'''

self.degree = len(coeffs)

self.coeffs = [float(c) for c in coeffs]

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

'''Converts to string representation'''

out = ''

for i, c in enumerate(self.coeffs):

if c == 0.0:

continue

out += self.to\_str(c, i)

return out

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

'''Printing at the terminal'''

return 'Poly({})'.format(self.coeffs)

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

'''Fetches the coeffiecent that has degree k'''

try:

if 0 <= k <= self.degree:

return self.coeffs[k]

else:

raise ValueError()

except ValueError:

print('Index out of range or non-integer')

def \_\_add\_\_(self, poly2):

'''Adds 2 polynomials Returns Poly'''

upto = len(self.coeffs)

result = []

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

result.append(self.coeffs[i] + poly2[i])

for i in range(upto, len(poly2.coeffs)):

result.append(poly2[i])

return Poly(result)

def \_\_mul\_\_(self, poly2):

'''Multiplies 2 polynomials Returns Poly'''

terms = {}

for i, c in enumerate(self.coeffs):

for j, c2 in enumerate(poly2.coeffs):

terms[i+j] = terms.get(i+j, 0) + c\*c2

return Poly([terms[sums] for sums in terms.keys()])

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

'''Scalar multiplication. Returns Poly'''

return Poly([k \* self.coeffs[c] for c in range(len(self.coeffs))])

def \_\_eq\_\_(self, poly2):

'''Returns True if 2 polynomials are equal'''

if self.coeffs == poly2.coeffs:

return True

else:

return False

def \_\_ne\_\_(self, poly2):

'''Return True if 2 polynomials are not equal'''

if self.\_\_eq\_\_(poly2):

return False

else:

return True

def eval(self, x):

'''Computes Polynomial value'''

sum = 0

for i, c in enumerate(self.coeffs):

if i == 0:

sum += c

if i >= 1:

sum += c \* (x\*\*i)

return sum

def to\_str(self, c, i):

'''makes given term a string'''

cstr = ''

if i != 0:

cstr += ' + '

if i == 0:

cstr += '{}'.format(str(c))

elif i == 1:

cstr += '{}x'.format(str(c))

elif i > 1:

cstr += '{}x^{}'.format(str(c), str(i))

return cstr

def TestPoly():

def unittest(b, tname):

result = 'FAILED'

if b:

result = 'PASSED'

print('{:<15} : {:<10}'.format(tname, result))

coeff = [1, 2, 3, 4]

coef2 = [5, 6, 7, 8]

passed = [1.0, 2.0, 3.0, 4.0]

p = Poly(coeff)

p2 = Poly(coef2)

# Poly.init

unittest(p.coeffs == passed, '\_\_init\_\_ ')

# test str

pstr = str(p)

unittest(pstr == '1.0 + 2.0x + 3.0x^2 + 4.0x^3', '\_\_str\_\_ ')

# test repr

prepr= repr(p)

unittest(prepr == 'Poly([1.0, 2.0, 3.0, 4.0])', '\_\_repr\_\_ ')

# test getitem

pget = p.\_\_getitem\_\_(2)

unittest(pget == 3.0, '\_\_getitem\_\_ ')

#test add

padd = p + p2

unittest(str(padd) == '6.0 + 8.0x + 10.0x^2 + 12.0x^3', '\_\_add\_\_' )

# test mul

pmul= p \* p2

unittest(str(pmul) == '5.0 + 16.0x + 34.0x^2 + 60.0x^3 + 61.0x^4 + 52.0x^5 + 32.0x^6', '\_\_mult\_\_')

# test rmul

prmul= 3 \* p

unittest(str(prmul) == '3.0 + 6.0x + 9.0x^2 + 12.0x^3', '\_\_rmult\_\_')

# test eq

peq = Poly(coeff)

unittest(p == peq, '\_\_eq\_\_ ')

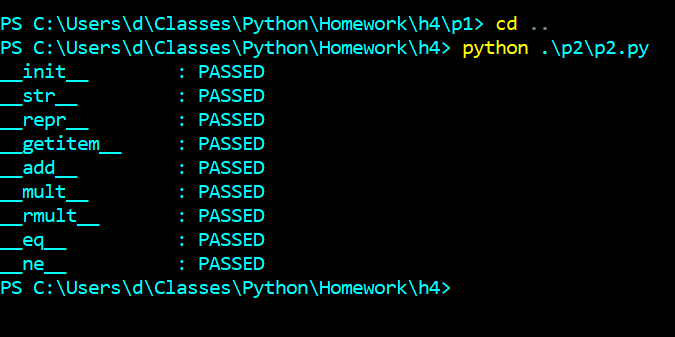
# test neq

pne = Poly(p.coeffs)

unittest(p2 != pne, '\_\_ne\_\_ ')

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

TestPoly()



Problem 3. HR Classes

# -\*- coding: utf-8 -\*-

class Employee(object):

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

self.\_name = name

self.\_phone = phone

self.\_sal = float(sal)

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

return '\n{}: {}; phone: {}; sal: {:<.2f}'.format(

self.\_\_class\_\_.\_\_name\_\_,

self.\_name,

self.\_phone,

self.sal\_total())

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

return "\n{}('{}', '{}', {:<.2f})".format(

self.\_\_class\_\_.\_\_name\_\_,

self.\_name,

self.\_phone,

self.sal\_total())

def sal\_total(self):

'''Returns the total salary'''

return self.\_sal

def name(self):

'''Returns a string'''

return self.\_name

def phone(self):

'''Returns a string'''

return self.\_phone

def sal(self):

return self.\_sal

class Manager(Employee):

'''Employee that has a bonus'''

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

Employee.\_\_init\_\_(self, name, phone, sal)

self.\_bonus = float(bonus)

def sal\_total(self):

return self.\_sal + self.\_bonus

class Ceo(Manager):

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

Manager.\_\_init\_\_(self, name, phone, sal, bonus)

self.\_stock = float(stock)

def sal\_total(self):

return self.\_sal + self.\_stock + self.\_bonus

class Engineer(Employee):

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

Employee.\_\_init\_\_(self, name, phone, sal)

def print\_staff(staff):

for s in staff:

print(repr(s))

def list\_staff():

en = Engineer('Wilbert', '4567890', 70000)

en2 = Engineer('Doug', '7891234', 70000)

m = Manager('Phil', '1233456', 50000, 10000)

em = Employee('Pam', '1234567', 60000)

em2 = Employee('Randy', '1233333', 55000)

c = Ceo('Hojo', '1234444', 800000, 10000, 400000)

return [en, m, em, c, en2, em2]

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

print\_staff(list\_staff())

