Homework 4

Problem 1. Top Movies and Actors

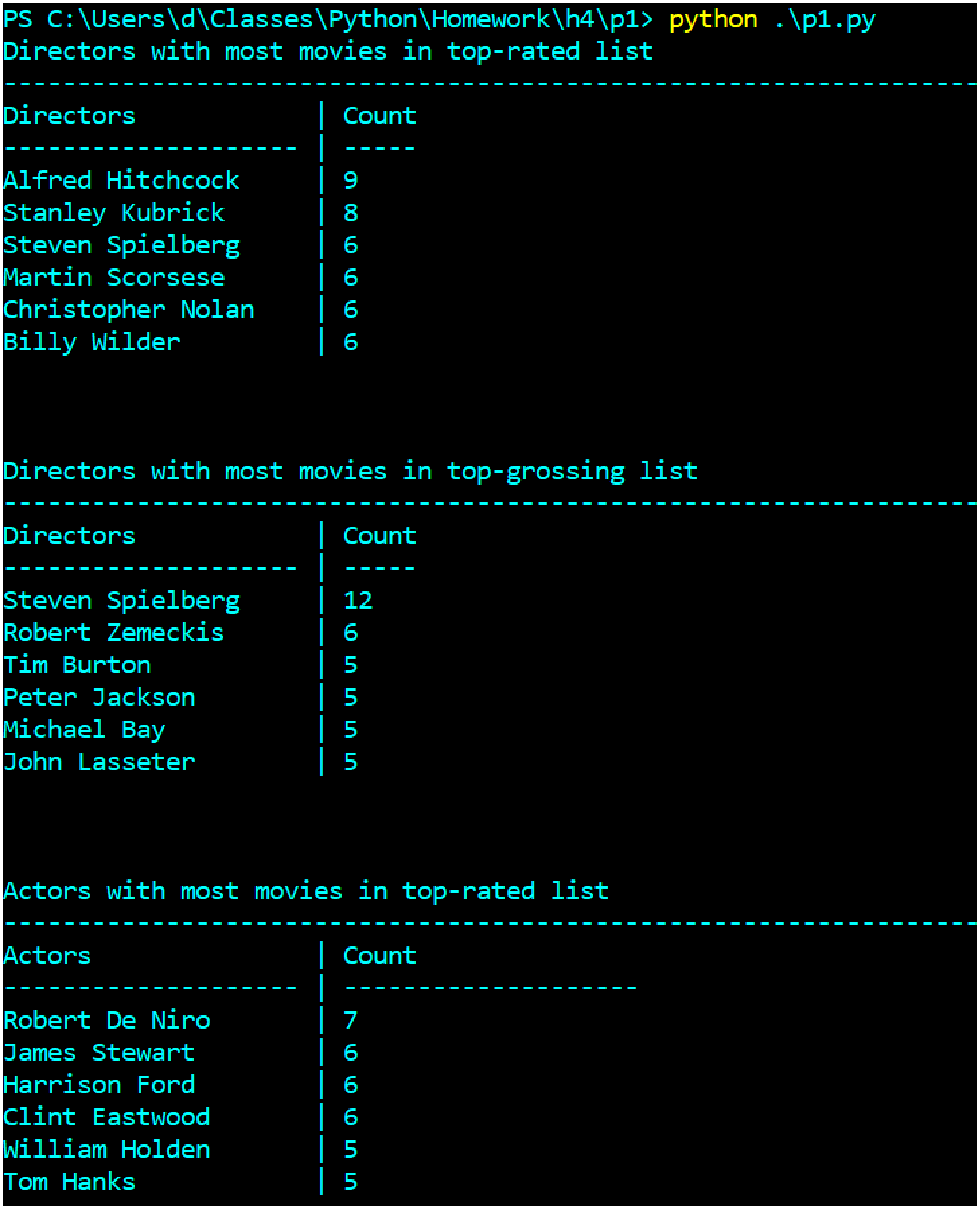
Top-Down Design:

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

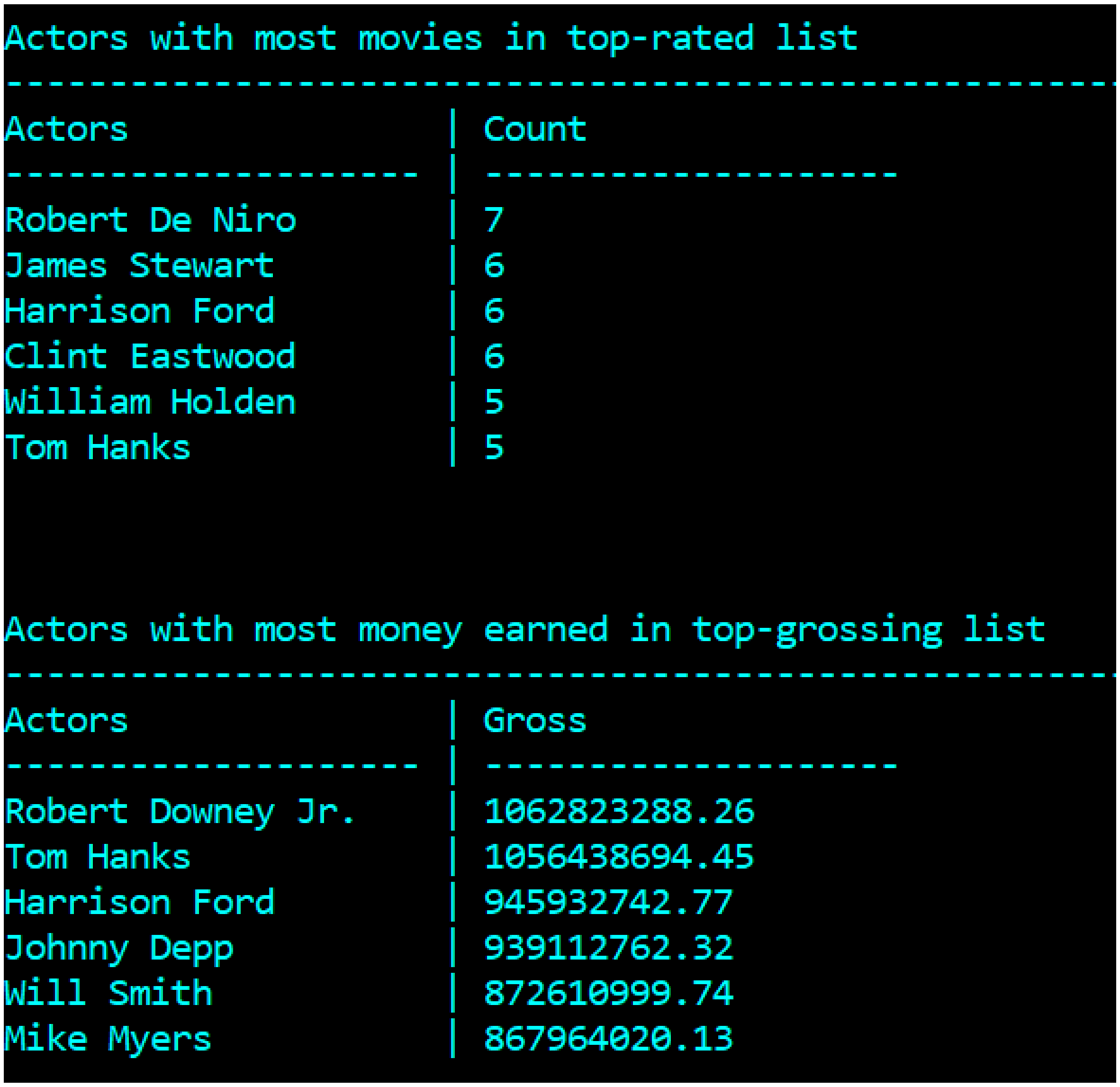
Shared Functionality:

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.

```python
# -*- 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()
```



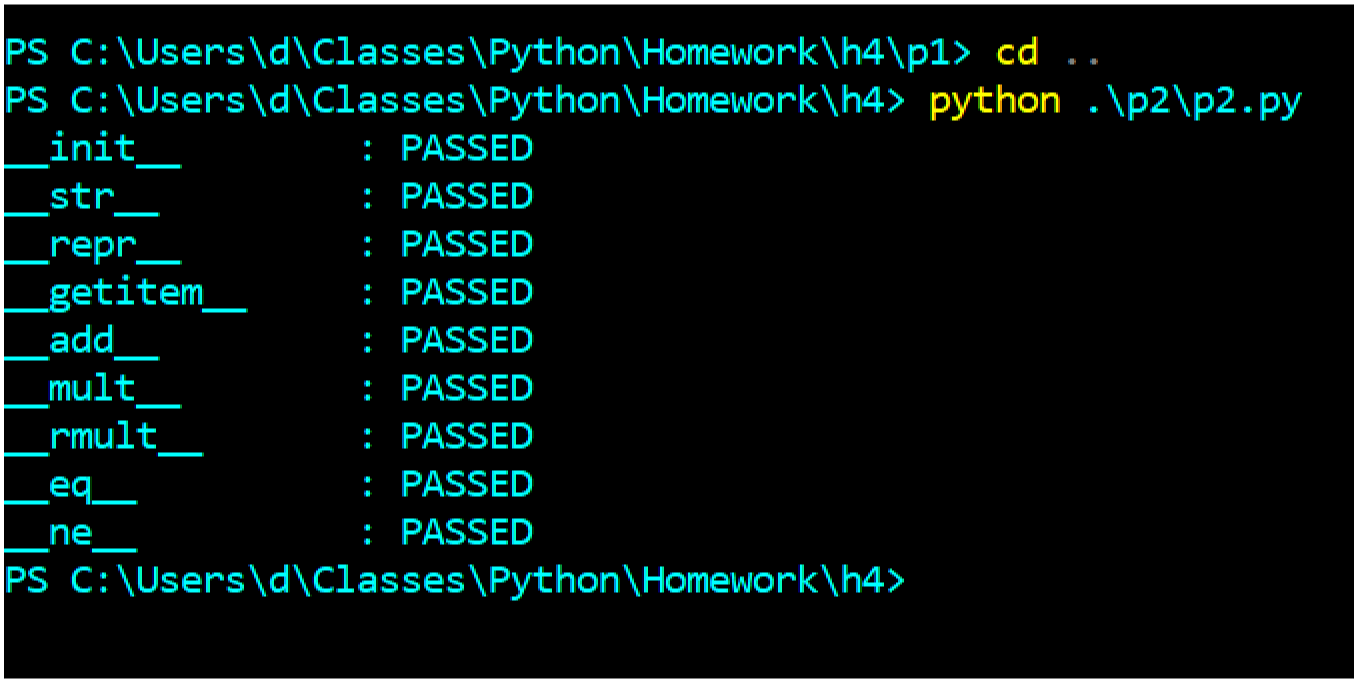
```
PS C:\Users\d\Classes\Python\Homework\h4\p1> python .\p1.py
Directors with most movies in top-rated list
------------------------------------------------------------------
Directors            | Count
-------------------- | -----
Alfred Hitchcock     | 9
Stanley Kubrick      | 8
Steven Spielberg     | 6
Martin Scorsese      | 6
Christopher Nolan    | 6
Billy Wilder         | 6



Directors with most movies in top-grossing list
------------------------------------------------------------------
Directors            | Count
-------------------- | -----
Steven Spielberg     | 12
Robert Zemeckis      | 6
Tim Burton           | 5
Peter Jackson        | 5
Michael Bay          | 5
John Lasseter        | 5



Actors with most movies in top-rated list
------------------------------------------------------------------
Actors               | Count
-------------------- | --------------------
Robert De Niro       | 7
James Stewart        | 6
Harrison Ford        | 6
Clint Eastwood       | 6
William Holden       | 5
Tom Hanks            | 5
```



```
Actors with most movies in top-rated list
-----------------------------------------------------
Actors                 | Count
---------------------  | ---------------------
Robert De Niro         | 7
James Stewart          | 6
Harrison Ford          | 6
Clint Eastwood         | 6
William Holden         | 5
Tom Hanks              | 5


Actors with most money earned in top-grossing list
-----------------------------------------------------
Actors                 | Gross
---------------------  | ---------------------
Robert Downey Jr.      | 1062823288.26
Tom Hanks              | 1056438694.45
Harrison Ford          | 945932742.77
Johnny Depp            | 939112762.32
Will Smith             | 872610999.74
Mike Myers             | 867964020.13
```

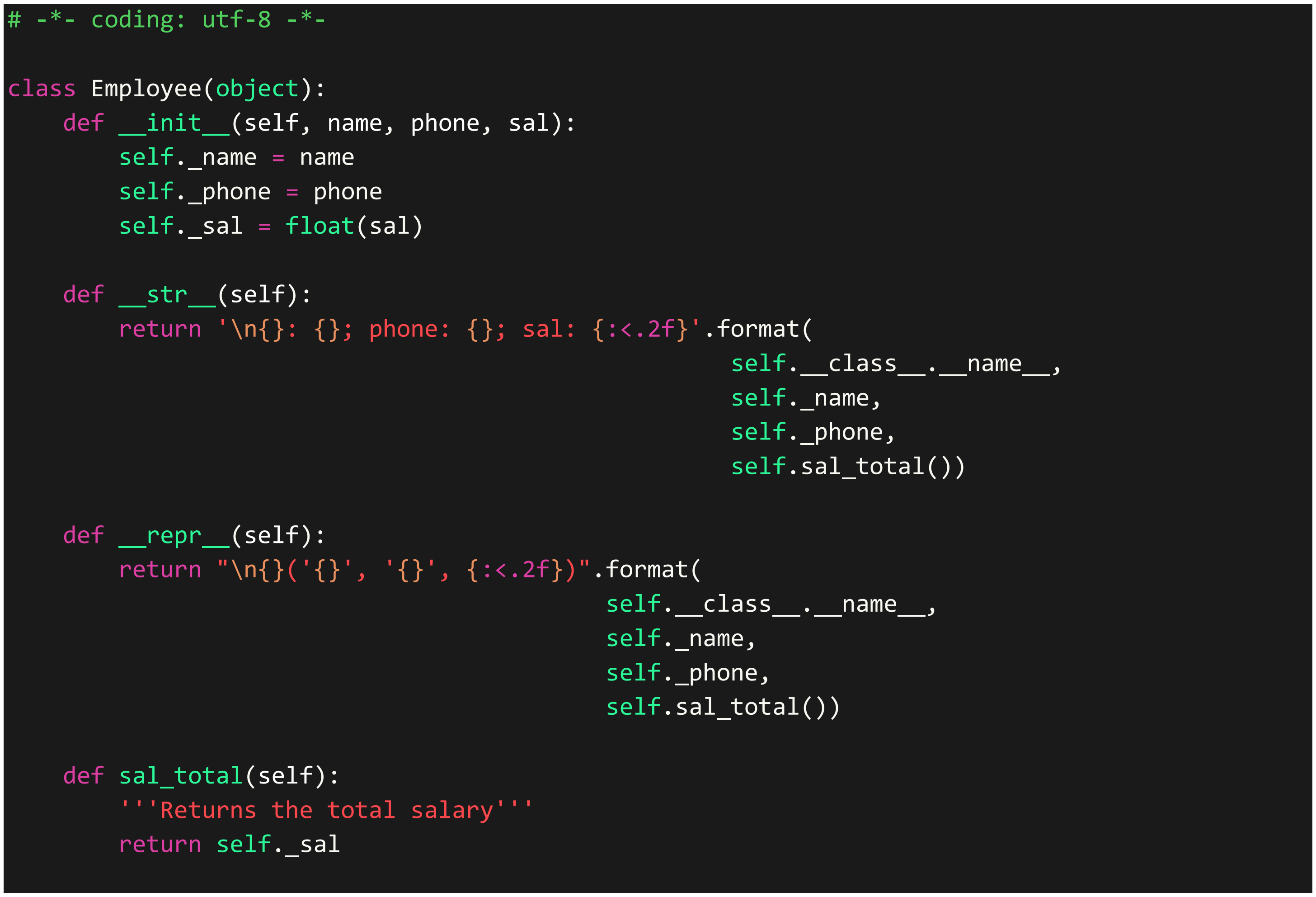
Problem 2. Polynomial Class

```python
# -*- 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()
```

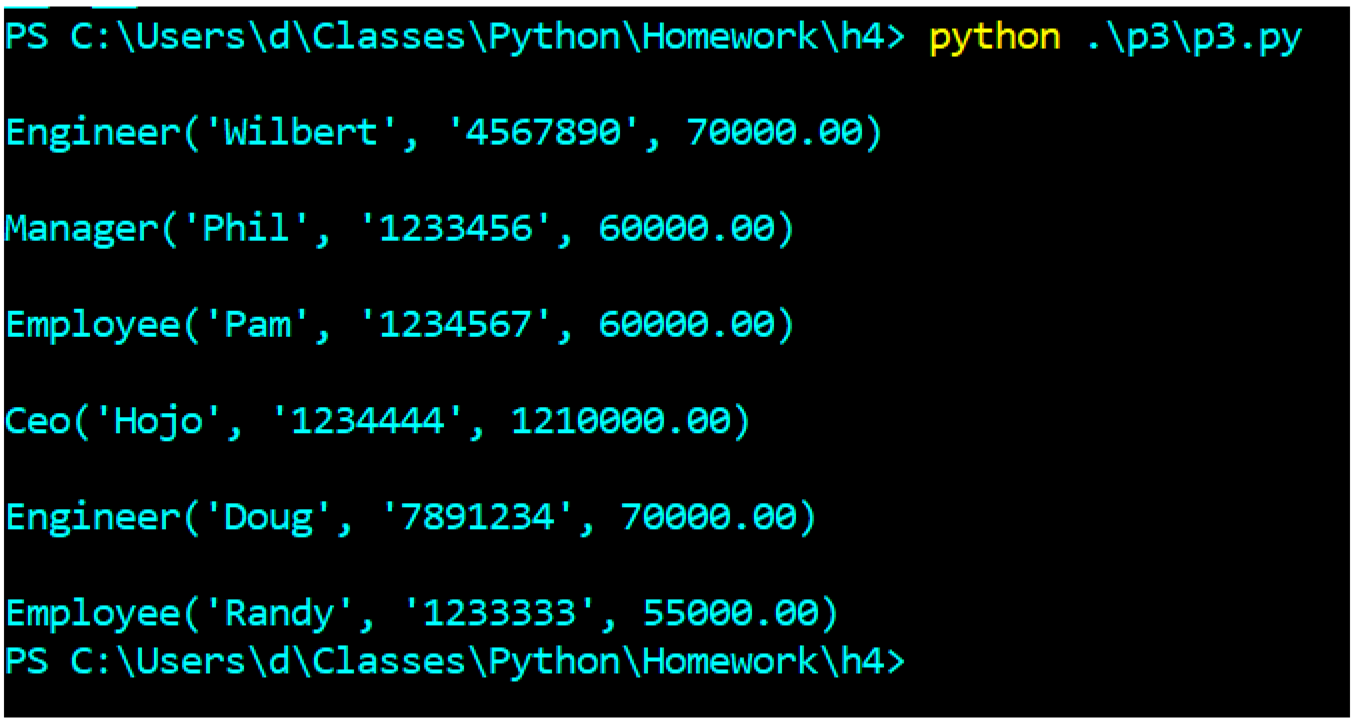


```
PS C:\Users\d\Classes\Python\Homework\h4\p1> cd ..
PS C:\Users\d\Classes\Python\Homework\h4> python .\p2\p2.py
__init__        : PASSED
__str__         : PASSED
__repr__        : PASSED
__getitem__     : PASSED
__add__         : PASSED
__mult__        : PASSED
__rmult__       : PASSED
__eq__          : PASSED
__ne__          : PASSED
PS C:\Users\d\Classes\Python\Homework\h4>
```

Problem 3. HR Classes



```python
# -*- 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())
```



```
PS C:\Users\d\Classes\Python\Homework\h4> python .\p3\p3.py

Engineer('Wilbert', '4567890', 70000.00)

Manager('Phil', '1233456', 60000.00)

Employee('Pam', '1234567', 60000.00)

Ceo('Hojo', '1234444', 1210000.00)

Engineer('Doug', '7891234', 70000.00)

Employee('Randy', '1233333', 55000.00)
PS C:\Users\d\Classes\Python\Homework\h4>
```