**[Assignment 9](https://mymasonportal.gmu.edu/webapps/assignment/uploadAssignment?content_id=_7735626_1&course_id=_331470_1&group_id=&mode=view)**

**Description:**

Python Data and Computations in the book, Computational Methods for Bioinformatics: Python 3.4, Third Edition by Jason Kinser.

Part 1

All problems for Chapter 16

Part 2

All worksheet exercises for learning module 23 Python SQL

Part 3

All worksheet exercises for learning module 24 Python Functions

This is an individual assignment and should be your own work.

**Format:** Please submit the assignment in pdf or MSWord compatible format. Please show code used and output of the code in your submission.

**Point value:**

Part 1) 18 points (3 points per problem)

Part 2) 6 points (3 points per problem)

Part 3) 16 points total ( 2 points per problem for worksheets 1 & 4, 4 points per problem for worksheets 2 & 3)

**Due Date:** April 9, 2018

16.5 Problems

1. Create a function named Aper that receives a single argument named indata. This function should print to the console the string “The input is: ” followed by the value of indata.

def Aper(indata):

print('The input is', indata)

Aper('indata')

*The input is indata*

2. Create a function like the previous but it prints the value of indata three times, each on a separate line.

def Aper(indata):

for i in range(3):

print('The input is', indata)

Aper('indata')

*The input is indata*

*The input is indata*

*The input is indata*

3. Create a function named Larger which receives two arguments, adata and bdata. The function should return the larger of the two values.

def Larger(adata, bdata):

if adata > bdata:

print(adata)

else:

print(bdata)

Larger(5,9)

4. Create a function named Complement that receives a DNA string and returns the complement of that string.

def Larger(adata, bdata):

if adata > bdata:

return adata

else:

return bdata

val = Larger(5,9)

print(val)

*9*

5. Create a function that has as its argument a default filename (such as the file for Romeo and Juliet). The function should return the length of the file (number of characters in the file). Run the function again with a different filename which overrides the default filename.

def question5(fname='romeo-and-juliet.txt'):

fvar = open(fname, 'r')

fvar2 = fvar.read()

print(len(fvar2))

question5("hw9.py")

*467*

6. Create two functions. The first is BF which receives a string and converts the letters to all capitals. The second is BA which receives a string and removes the spaces. Then it passes that string to BF and receives the result. The function BA should then return the resultant string which should be all caps and have no spaces.

def BF(indata):

return indata.upper()

def BA(indata2):

indata2 = BF(indata2)

return indata2.replace(" ", "")

print(BA("yell ow"))

*YELLOW*

7. Create the function CountryFromCndx which receives a cndx value and returns the name of the associated country.

def LoadCountry(myfile="country.csv"):

txt = open(myfile).read().split ("\n")

actors = []

for i in range(1, len(txt)-1):

row = txt[i].split (",")

actors.append((int(row[0]), row[1]))

return actors

def LoadInCountry(myfile="incountry.csv"):

txt = open(myfile).read().split ("\n")

actors = []

for i in range(1, len(txt)-1):

row = txt[i].split (",")

actors.append((int(row[0]), int(row[1]), int(row[2])))

return actors

def CountryFromCndx(cndx):

countrys = LoadCountry()

country = ()

incountrys = LoadInCountry()

incntry = ()

for cntry in incountrys:

if cntry[0] == int(cndx):

incntry = cntry

break

for cntry in countrys:

if cntry[0] == incntry[2]:

return cntry[1]

print(CountryFromCndx(21))

*UK*

8. Create a function named TitleFromGrade which receives an input grade and returns a list of movie titles for the movies that have that grade.

def LoadMovies(myfile="movies800.tsv"):

txt = open(myfile).read().split ("\n")

actors = []

for i in range(1, len(txt)-1):

row = txt[i].split ("\t")

actors.append((int(row[0]), row[1], row[2], row[3]))

return actors

def TitleFromGrade(grade):

movierows = LoadMovies()

movies = []

for movie in movierows:

if str(movie[3]) == str(grade):

movies.append(movie[1])

return movies

print(TitleFromGrade(10))

*['An American Tail: Fievel Goes West', 'Big Bad Love', 'Big Fish', 'Cannery Row', 'Cautiva', 'Fargo', 'Grand Hotel', 'Holes', 'Munich', '"O Brother, Where Are Thou"', 'Rope', 'The Fugitive', 'The Great Debaters', 'The Illusionist', 'The Third Man', 'Young Frankenstein', 'Once Upon a Time in the West', 'Joyeux Noel', 'Rebecca', 'Rear Window', 'The Sting', 'Harold and Maude', 'Million Dollar Baby', 'Citizen Kane', 'La Vie En Rose', 'The Sensation of Sight', 'Under the Bombs', 'The Lives of Others', 'M', '12 Angry Men', 'Key Largo', 'Kolya', 'My Sweet Little Village', 'The Dinner Game', 'Zift']*

9. Create a function named TitlesFromGrades. This function should return a list of movie names which have specified grades. The input to the function is a list of the specified grades. This function should iteratively call TitleFromGrade in the previous problem.

def TitlesFromGrades(grades):

movies = []

for grade in grades:

movies = movies + TitleFromGrade(grade)

return movies

print(TitlesFromGrades([1,10]))

*['I Want Candy', 'Romance & Cigarettes', 'Sex and Breakfast', 'The Day After Tomorrow', 'Tremors', 'What the #$\*! Do We Know?!', 'Broken Flowers', 'Monte Walsh', 'Body of Lies', 'Mamma Mia!', 'Summer Hours', 'Mallrats', 'Tenure', 'Metroland', 'Corked', 'Dick', 'An American Tail: Fievel Goes West', 'Big Bad Love', 'Big Fish', 'Cannery Row', 'Cautiva', 'Fargo', 'Grand Hotel', 'Holes', 'Munich', '"O Brother, Where Are Thou"', 'Rope', 'The Fugitive', 'The Great Debaters', 'The Illusionist', 'The Third Man', 'Young Frankenstein', 'Once Upon a Time in the West', 'Joyeux Noel', 'Rebecca', 'Rear Window', 'The Sting', 'Harold and Maude', 'Million Dollar Baby', 'Citizen Kane', 'La Vie En Rose', 'The Sensation of Sight',*

*'Under the Bombs', 'The Lives of Others', 'M', '12 Angry Men', 'Key Largo', 'Kolya', 'My Sweet Little Village', 'The Dinner Game', 'Zift']*

10. Create a function that returns the average and standard deviation of the length of movie titles.

import statistics

def TitleList():

movies = LoadMovies()

titles = []

for mov in movies:

titles.append(mov[1])

return titles

def AvgStdev(titles):

titlelengths = []

for tit in titles:

titlelengths.append(len(tit))

return statistics.mean(titlelengths), statistics.pstdev(titlelengths)

print("avg, standard deviation")

print(AvgStdev(TitleList()))

*avg, standard deviation*

*(15.195, 8.192647618444235)*

11. Create a function that returns the first names of the actors who have a given last name, for example ”Keaton”.

def LoadActors(myfile="actors.tsv"):

txt = open(myfile).read().split ("\n")

actors = []

for i in range(1, len(txt)-1):

row = txt[i].split ("\t")

actors.append((int(row[0]), row[1], row[2]))

return actors

def actornames(last):

actorlist = LoadActors()

firstnames = []

for act in actorlist:

if act[2] == last:

firstnames.append(act[1])

return firstnames

print(actornames("Smith"))

*['Will', 'Maggie']*

12. Create a function that returns the name of all actors that have identical first initials.

def matchinginital(firstname, lastname):

matches = []

actorlist = LoadActors()

for actor in actorlist:

if firstname[0:1] == actor[1][0:1] and lastname[0:1] == actor[2][0:1]:

matches.append(actor)

return matches

def matchinginitials():

actorlist = LoadActors()

matches = []

for actor in actorlist:

match = matchinginital(actor[1], actor[2])

if len(match) > 1:

matches.append(match)

return matches

print(matchinginitials())

*[[(2, 'Andy', 'Griffith'), (115, 'Ava', 'Gardner'), (145, 'Andy', 'Garcia'), (276, 'Alec', 'Guiness'), (588, 'Alice', 'Ghostley')], [(3, 'Walter', 'Matthau'), (66, 'William H.', 'Macy')], [(4, 'Orson', 'Welles'), (649, 'Owen', 'Wilson')], [(5, 'F. Murray', 'Abraham'), (558, 'Fred', 'Astaire')], [(6, 'James (Jimmy)', 'Stewart'), (134, 'John', 'Spencer'), (242, 'Julia', 'Stiles'), (590, 'Jean', 'Stapleton'), (707, 'J. K.', 'Simmons')], [(7, 'Dom', 'DeLuise'), (135, 'Danny', 'DeVito'), (546, 'Doris', 'Day')], [(8, 'John', 'Belushi'), (39, 'Juliette', 'Binochet'), (89, 'Jeff', 'Bridges'), (94, 'Jack', 'Black'), (218, 'Joe Don', 'Baker'), (374, 'Jack', 'Benny'), (443, 'Jim', 'Brown'), (455, 'James', 'Brown'), (480,*

*'James', 'Belushi'), (539, 'Jaqueline', 'Bisset')], [(9, 'Tom', 'Hanks'), (156, 'Timothy', 'Hutton'), (490, 'Tom', 'Hulce'), (542, 'Tippi', 'Hedren')], [(10, 'James', 'Garner'), (36, 'Jeff', 'Goldblum'), (105, 'John', 'Goodman'), (191, 'Janeane', 'Garofalo'), (357, 'John', 'Gielgud'), (506, 'Johanna', 'Gleeson'), (728, 'Jasmine', 'Guy')], [(11, 'Michael J.', 'Fox'), (64, 'Morgan', 'Freeman'), (250, 'Marty', 'Feldman'), (502, 'Mia', 'Farrow'), (741, 'Meg', 'Foster')], [(12, 'Christopher', 'Lloyd'), (72, 'Cloris', 'Leachman'), (271, 'Christopher', 'Lee'), (497, 'Charles', 'Laughton'), (642, 'Carole', 'Lombard')], [(13, 'Johnny', 'Depp'), (219, 'Judi', 'Dench'), (373, 'Jimmy', 'Durante'), (551, 'John', 'Denver'),*

*(672, 'Jeff', 'Daniels'), (740, 'Jamel', 'Debbouze')], [(14, 'Aidan', 'Quinn'), (168, 'Anthony', 'Quinn')], [(15, 'Julieanne', 'Moore'), (46, 'John', 'Malkovich'), (52, 'John', 'Mahoney'), (112, 'James', 'Mason')], [(16, 'Arliss', 'Howard'), (103, 'Anothy', 'Hopkins'), (278, 'Audrey', 'Hepburn'), (351, 'Anjelica', 'Huston'), (530, 'Alfred', 'Hitchcock'), (553, 'Anne', 'Heche')], [(17, 'Debra', 'Winger'), (53, 'Dianne', 'Wiest'), (60, 'Dennis', 'Weaver'), (157, 'Denzel', 'Washington'), (494, 'Damon', 'Wayans')], [(18, 'Tim', 'Burton'), (323, 'Terry', 'Bradshaw'), (688, 'Tom', 'Berenger')], [(19, 'John', 'Turturro'), (44, 'Jessica', 'Tandy'), (154, 'John', 'Travolta'), (385, 'Janine', 'Turner'), (395, 'Jennifer', 'Tilly'), (464, 'John', 'Thaw')], [(20, 'Bill', 'Pullman'), (104, 'Brad', 'Pitt'), (198, 'Bill', 'Paxton'), (263, 'Bernadette', 'Peters')], [(21, 'Chris', 'Pine'), (333, 'Christopher', 'Plummer')], [(22, 'John', 'Candy'), (50, 'John', 'Cusack'), (80, 'Jim', 'Carrey'), (96, 'Jackie', 'Chan'), (124, 'John', 'Cleese'), (155, 'James', 'Cromwell'), (160, 'James', 'Coburn'), (184, 'Joseph', 'Cotton'), (264, 'James', 'Caan'), (292, 'Jamie Lee', 'Curtis'), (331, 'Joan', 'Cusack'), (334, 'Jennifer', 'Connelly'), (397, 'Jane', 'Curtin'), (407, 'Jackie', 'Cooper'), (496, 'James', 'Cagney'), (507, 'John', 'Carradine'), (738, 'Jim', 'Carter')], [(23, 'Alan', 'Alda'), (182, 'Alan', 'Arkin'), (577, 'Adam', 'Arkin')], [(24, 'Rhea', 'Perlman'), (193, 'Ron', 'Perlman'), (202, 'Robert', 'Preston'), (440, 'Richard', 'Pryor'), (621, 'Robert', 'Prosky')], [(26, 'Dan', 'Aykroyd'), (40, 'Don', 'Ameche'), (501, 'Danny', 'Aiello')], [(29, 'Frank', 'McRae'), (230, 'Frances', 'McDormand'), (478, 'Fred', 'MacMurray')], [(30, 'John', 'Huston'), (751, 'John', 'Houseman')], [(31, 'Paul', 'Newman'), (722, 'Patricia', 'Neal')], [(32, 'Tony', 'Shaloub'), (435, 'Telly', 'Savalas'), (481, 'Tom', 'Selleck')], [(33, 'George', 'Carlin'), (126, 'George', 'Clooney'), (284, 'Glenn', 'Close')], [(34, 'John', 'Ratzenberger'), (61, 'Julia', 'Roberts'), (68, 'Jean', 'Reno'), (205, 'John', 'Rhys-Davies'), (268, 'Jason', 'Robards'), (280, 'John', 'Ritter'), (281,*

*'John C.', 'Reilly'), (344, 'Joan', 'Rivers'), (610, 'Jack', 'Riley'), (647, 'Judge', 'Reinhold')], [(10, 'James', 'Garner'), (36, 'Jeff', 'Goldblum'), (105, 'John', 'Goodman'), (191, 'Janeane', 'Garofalo'), (357, 'John', 'Gielgud'), (506, 'Johanna', 'Gleeson'), (728, 'Jasmine', 'Guy')], [(38, 'Richard', 'Gere'), (272, 'Rachel', 'Griffiths'), (318, 'Robert', 'Guillaume'), (413, 'Ruth', 'Gordon'), (476, 'Ray', 'Gould'), (720, 'Rupert', 'Graves'), (721, 'Ron', 'Glass')], [(8, 'John', 'Belushi'), (39, 'Juliette', 'Binochet'), (89, 'Jeff', 'Bridges'), (94, 'Jack', 'Black'), (218, 'Joe Don', 'Baker'), (374, 'Jack', 'Benny'), (443, 'Jim', 'Brown'), (455, 'James', 'Brown'), (480, 'James', 'Belushi'), (539, 'Jaqueline', 'Bisset')], [(26, 'Dan', 'Aykroyd'), (40, 'Don', 'Ameche'), (501, 'Danny', 'Aiello')], [(41, 'Wilford', 'Brimley'), (422, 'Warren', 'Beatty'), (515, 'Walter', 'Brennan'), (559, 'William', 'Baldwin')], [(42, 'Hume', 'Cronyn'), (316, 'Helen Bonham', 'Carter'), (581, 'Harry', 'Connick Jr.')], [(43, 'Brian', 'Dennehy'), (495, 'Bette', 'Davis'), (528, 'Bruce', 'Dern')], [(19, 'John', 'Turturro'), (44, 'Jessica', 'Tandy'), (154, 'John', 'Travolta'), (385, 'Janine', 'Turner'), (395, 'Jennifer', 'Tilly'), (464, 'John', 'Thaw')], [(45, 'Nicholas', 'Cage'), (356, 'Noel', 'Coward')], [(15, 'Julieanne', 'Moore'), (46, 'John', 'Malkovich'), (52, 'John', 'Mahoney'), (112, 'James', 'Mason')], [(47, 'Ving', 'Rhames'), (82, 'Vanessa', 'Redgrave')], [(48, 'Steve', 'Buscemi'), (671, 'Sandra', 'Bullock')], [(49, 'Colm', 'Meaney'), (462, 'Cheech', 'Marin')], [(22, 'John', 'Candy'), (50, 'John', 'Cusack'), (80, 'Jim', 'Carrey'), (96, 'Jackie', 'Chan'), (124, 'John', 'Cleese'), (155, 'James', 'Cromwell'), (160, 'James', 'Coburn'), (184, 'Joseph', 'Cotton'), (264, 'James', 'Caan'), (292, 'Jamie Lee', 'Curtis'), (331, 'Joan', 'Cusack'), (334, 'Jennifer', 'Connelly'), (397, 'Jane', 'Curtin'), (407, 'Jackie', 'Cooper'), (496, 'James', 'Cagney'), (507, 'John', 'Carradine'), (738, 'Jim', 'Carter')], [(51, 'Steve', 'Carrell'), (63, 'Sean', 'Connery'), (197, 'Steve', 'Coogan'), (261, 'Sid', 'Ceasar'), (485, 'Sid', 'Caesar'), (594, 'Sebastian', 'Cabot')], [(15, 'Julieanne', 'Moore'), (46, 'John', 'Malkovich'), (52, 'John', 'Mahoney'), (112, 'James', 'Mason')], [(17, 'Debra', 'Winger'), (53, 'Dianne', 'Wiest'), (60, 'Dennis', 'Weaver'), (157, 'Denzel', 'Washington'), (494, 'Damon', 'Wayans')], [(54, 'Pierce', 'Brosnan'), (249, 'Peter', 'Boyle')], [(55, 'Linda', 'Hamilton'), (655, 'Larry', 'Hagman'), (733, 'Lucas', 'Haas')], [(57, 'Julie', 'Kavner'), (181, 'Johnny', 'Knoxville'), (627, 'Jack', 'Klugman')], [(58, 'Michael', 'Caine'), (110, 'Maury', 'Chaykin'), (510, 'Marion', 'Cotillard')], [(17, 'Debra', 'Winger'), (53, 'Dianne', 'Wiest'), (60, 'Dennis', 'Weaver'), (157, 'Denzel', 'Washington'), (494, 'Damon', 'Wayans')], [(34, 'John', 'Ratzenberger'), (61, 'Julia', 'Roberts'), (68, 'Jean', 'Reno'), (205, 'John', 'Rhys-Davies'), (268, 'Jason', 'Robards'), (280, 'John', 'Ritter'), (281, 'John C.', 'Reilly'), (344, 'Joan', 'Rivers'), (610, 'Jack', 'Riley'), (647, 'Judge', 'Reinhold')], [(62, 'Clive', 'Owen'), (303, 'Catherine ', "O'Hara"), (597, 'Carroll', "O'Connor")], [(51, 'Steve', 'Carrell'), (63, 'Sean', 'Connery'), (197, 'Steve', 'Coogan'), (261, 'Sid', 'Ceasar'), (485, 'Sid', 'Caesar'), (594, 'Sebastian', 'Cabot')], [(11, 'Michael J.', 'Fox'), (64, 'Morgan', 'Freeman'), (250, 'Marty', 'Feldman'), (502, 'Mia', 'Farrow'), (741, 'Meg', 'Foster')], [(65, 'Goldie', 'Hawn'), (399, 'Gene', 'Hackman'), (484, 'Gregory', 'Hines')], [(3, 'Walter', 'Matthau'), (66, 'William H.', 'Macy')], [(67, 'Chevy', 'Chase'), (83, 'Charlotte', 'Church'), (86, 'Chris', 'Cooper'), (456, 'Cab', 'Calloway'), (468, 'Charles', 'Chaplin'), (658, 'Claudia', 'Cardinale')], [(34, 'John', 'Ratzenberger'), (61, 'Julia', 'Roberts'), (68, 'Jean', 'Reno'), (205, 'John', 'Rhys-Davies'), (268, 'Jason', 'Robards'), (280, 'John', 'Ritter'), (281, 'John C.', 'Reilly'), (344, 'Joan', 'Rivers'), (610, 'Jack', 'Riley'), (647, 'Judge', 'Reinhold')], [(69, 'Kevin', 'Costner'), (376, 'Kim', 'Crosby')], [(70, 'Minnie', 'Driver'), (138, 'Michael', 'Douglas'), (146, 'Matt', 'Damon'), (361, 'Marlene', 'Dietrich'), (447, 'Mos', 'Def')], [(71, 'Mel', 'Brooks'), (150, 'Matthew', 'Broderick'), (151, 'Marlon', 'Brando'), (348, 'Mel', 'Blanc'), (364, 'Milton', 'Berle')], [(12, 'Christopher', 'Lloyd'), (72, 'Cloris', 'Leachman'), (271, 'Christopher', 'Lee'), (497, 'Charles', 'Laughton'), (642, 'Carole', 'Lombard')], [(73, 'Madelaine', 'Kahn'), (297, 'Michael', 'Keaton'), (409, 'Margot', 'Kidder'), (465,*

*'Moira', 'Kelly')], [(74, 'Harvey', 'Kormann'), (335, 'Harvey', 'Keitel')], [(76, 'Sigourney', 'Weaver'), (654, 'Shelley', 'Winters')], [(77, 'Jon', 'Voight'), (377, 'Jan-Michael', 'Vincent'), (756, 'Jim', 'Varney')], [(79, 'Shia', 'LeBeouf'), (571, 'Sophia', 'Loren'), (691, 'Stan', 'Laurel')], [(22, 'John', 'Candy'), (50, 'John', 'Cusack'), (80, 'Jim', 'Carrey'), (96, 'Jackie', 'Chan'), (124, 'John', 'Cleese'), (155, 'James', 'Cromwell'), (160, 'James', 'Coburn'), (184, 'Joseph', 'Cotton'), (264, 'James', 'Caan'), (292, 'Jamie Lee', 'Curtis'), (331, 'Joan', 'Cusack'),*

*(334, 'Jennifer', 'Connelly'), (397, 'Jane', 'Curtin'), (407, 'Jackie', 'Cooper'), (496, 'James', 'Cagney'), (507, 'John', 'Carradine'), (738, 'Jim', 'Carter')], [(81, 'Carol', 'Burnett'), (159, 'Charles', 'Bronson'), (209, 'Cate', 'Blanchett')], [(47, 'Ving', 'Rhames'), (82, 'Vanessa',*

*'Redgrave')], [(67, 'Chevy', 'Chase'), (83, 'Charlotte', 'Church'), (86, 'Chris', 'Cooper'), (456, 'Cab', 'Calloway'), (468, 'Charles', 'Chaplin'), (658, 'Claudia', 'Cardinale')], [(84, 'Joss', 'Ackland'), (201, 'Julie', 'Andrews'), (580, 'Jennifer', 'Aniston'), (716, 'Judith', 'Anderson')], [(85, 'Kurt', 'Russell'), (291, 'Keanu', 'Reeves')], [(67, 'Chevy', 'Chase'), (83, 'Charlotte', 'Church'), (86, 'Chris', 'Cooper'), (456, 'Cab', 'Calloway'), (468, 'Charles', 'Chaplin'), (658, 'Claudia', 'Cardinale')], [(87, 'Amy', 'Smart'), (92, 'Arnold', 'Schwarzenegger')], [(88,*

*'Robert', 'Downey Jr.'),*

13. Starting with the script in Code 16.36, write a function that sorts the (average grade, year) tuples from the highest average grade to the lowest average grade.

def AverageGradeFromYear(movies, year):

movieyear = []

for movie in movies:

if movie[2]:

if str(movie[2]) == str(year):

movieyear.append(int(movie[3]))

return (statistics.mean(movieyear), year)

def AverageGradesByYear(movies):

years = []

for movie in movies:

years.append(movie[2])

years = set(years)

avgyear = []

for year in years:

avgyear.append(AverageGradeFromYear(movies, year))

avgyear.sort(key=lambda tup: tup[0], reverse=True)

return avgyear

movies = LoadMovies()

avgYearGrade = AverageGradesByYear(movies)

for tup in avgYearGrade:

print(tup)

*(10, '1931')*

*(10, '1948')*

*(10, '1932')*

*(9.5, '1957')*

*(9, '1965')*

*(9, '1928')*

*(8.5, '1954')*

*(8, '1934')*

*(8, '1947')*

*(8, '1936')*

*(8, '1971')*

*(8, '1945')*

*(8, '1973')*

*(7.75, '1970')*

*(7.666666666666667, '1941')*

*(7.5, '1972')*

*(7.5, '1959')*

*(7.5, '1935')*

*(7, '1933')*

*(7, '1967')*

*(7, '1958')*

*(7, '1946')*

*(7, '2014')*

*(7, '1951')*

*(7, '1944')*

*(6.875, '1968')*

*(6.833333333333333, '1975')*

*(6.75, '2000')*

*(6.666666666666667, '1956')*

*(6.657894736842105, '2006')*

*(6.555555555555555, '1996')*

*(6.526315789473684, '1994')*

*(6.5, '1938')*

*(6.5, '1940')*

*(6.5, '2010')*

*(6.454545454545454, '1982')*

*(6.416666666666667, '1997')*

*(6.384615384615385, '1991')*

*(6.333333333333333, '1949')*

*(6.3, '2001')*

*(6.25, '2011')*

*(6.25, '1966')*

*(6.230769230769231, '1990')*

*(6.111111111111111, '1998')*

*(6.071428571428571, '1983')*

*(6.044444444444444, '2008')*

*(6, '2017')*

*(6, '1955')*

*(6, '1969')*

*(6, '1974')*

*(6, '1929')*

*(5.9411764705882355, '1993')*

*(5.928571428571429, '1985')*

*(5.894736842105263, '1988')*

*(5.884615384615385, '1999')*

*(5.875, '1987')*

*(5.875, '2005')*

*(5.853658536585366, '2007')*

*(5.833333333333333, '1942')*

*(5.809523809523809, '2009')*

*(5.774193548387097, '2003')*

*(5.666666666666667, '1976')*

*(5.636363636363637, '1981')*

*(5.588235294117647, '1984')*

*(5.583333333333333, '2004')*

*(5.583333333333333, '1979')*

*(5.52, '2002')*

*(5.5, '1963')*

*(5.5, '1992')*

*(5.4, '2012')*

*(5.333333333333333, '1995')*

*(5.266666666666667, '1989')*

*(5.1, '1978')*

*(5, '1943')*

*(5, '1950')*

*(5, ' ')*

*(5, '1964')*

*(4.75, '1977')*

*(4.666666666666667, '1986')*

*(4.5, '1980')*

*(4.333333333333333, '1960')*

*(4, '1962')*

*(3, '1953')*

14. Write the necessary functions to return a list of movie mids for movies with a specified language.

def loadInLangs(myfile="inlangs.txt"):

txt = open(myfile).read().split ("\n")

actors = []

for i in range(1, len(txt)-1):

row = txt[i].split ("\t")

actors.append((int(row[0]), row[1], row[2]))

return actors

def loadLangs(myfile="langs.txt"):

txt = open(myfile).read().split ("\n")

langs = []

for i in range(1, len(txt)-1):

row = txt[i].split ("\t")

langs.append((int(row[0]), row[1]))

return langs

def GetLidFromLang(lang):

langs = loadLangs()

for l in langs:

if lang == l[1]:

return l[0]

def GetMidsByLid(lid):

movies = loadInLangs()

answr = []

for mov in movies:

if str(mov[2]) == str(lid):

answr.append(mov[0])

return answr

def GetMidsByLang(lang):

lid = GetLidFromLang(lang)

return GetMidsByLid(lid)

print(GetMidsByLang("Hebrew"))

*[87, 852]*

15. Write a function that uses MidsFromAid and AidFromNames that counts the number of movies for every actor. Sort this data from the most number of movies to the least. Print the five actors with the most movies from this sorted list.

import movies

def MoviesByActors():

isin = movies.LoadIsin("isin.txt")

actors = movies.LoadActors("actors.tsv")

actorcounts = []

for actor in actors:

fname = actor[1]

lname = actor[2]

aid = movies.AidFromName(actors, fname, lname)

num = len(movies.MidsFromAid(isin, aid))

actorcounts.append((fname, lname, num))

actorcounts.sort(key=lambda x: x[2], reverse=True)

return actorcounts

moviesbyactors = MoviesByActors()

for i in range(5):

print(moviesbyactors[i])

*('Alfred', 'Hitchcock', 23)*

*('Woody', 'Allen', 17)*

*('Dan', 'Aykroyd', 15)*

*('Steve', 'Martin', 15)*

*('Tom', 'Hanks', 13)*

Kinser, Jason. Computational Methods for Bioinformatics: Python 3.4 (Page 268). . Kindle Edition.