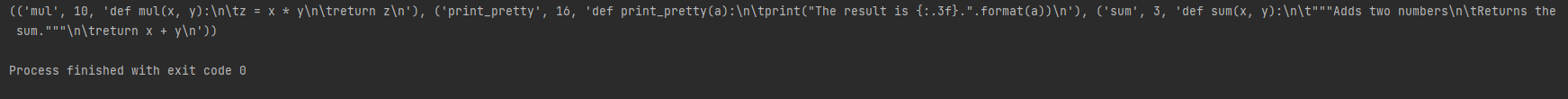
Python Programming

Homework 2

Jordan Diaz

Solution #1:

# Jordan Diaz, This program manipulates files and parses itself  
def line\_number(file\_name, write\_to\_this):  
 *""" This file takes the contents of one file and copies it into another file but with numbered lines """* # edge case  
 if file\_name == write\_to\_this:  
 print("Error: Cannot use the same file name")  
 exit(1)  
  
 try:  
 # Open the files  
 file\_name = open(file\_name, "r")  
 write\_to\_this = open(write\_to\_this, "w")  
  
 # Go through the file and write to the other  
 count = 1  
 for line in file\_name:  
 print(count, ". ", line, file=write\_to\_this, end="")  
 count += 1  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to read the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 finally:  
 # Close the files  
 file\_name.close()  
 write\_to\_this.close()  
  
  
def parse\_functions(file\_name):  
 try:  
 file\_name = open(file\_name, "r")  
  
 separation\_str = "def" + " "  
 curr\_function\_str = ""  
  
 count = 1  
 lst\_of\_def\_names = []  
 lst\_of\_positions = []  
  
 # Get the name of the function and the line number  
 for line in file\_name.readlines():  
  
 for n in range(0, len(line)):  
 avoid\_str = "#" + " "  
 if n == line.find(avoid\_str) or n + 1 == line.find(avoid\_str):  
 break  
 elif n == line.find(" " + "#"):  
 break  
 elif line.find(separation\_str) != -1 and line[n - 1] == ":":  
 curr\_function\_str += "\n"  
 elif line.find("def") == 0:  
 curr\_function\_str += line[n]  
 elif line.find(" ") == 0:  
 curr\_function\_str += line[n]  
  
 curr\_name = ""  
  
 # Build a list of positions and names of functions  
 if line.find("def") == 0:  
 for i in range(4, line.find("(")):  
 curr\_name += line[i]  
 lst\_of\_positions.append(count)  
 lst\_of\_def\_names.append(curr\_name)  
  
 count += 1  
  
 separation\_str = "def" + " "  
  
 lst\_of\_functions = curr\_function\_str.split(separation\_str)  
 lst\_of\_functions.pop(0)  
  
 for k in range(0, len(lst\_of\_functions)):  
 lst\_of\_functions[k] = separation\_str + lst\_of\_functions[k].replace(" ", "\t")  
  
 final\_lst = []  
 for m in range(0, len(lst\_of\_def\_names)):  
 tple = (lst\_of\_def\_names[m], lst\_of\_positions[m], lst\_of\_functions[m])  
 final\_lst.append(tple)  
  
 final\_lst.sort()  
 return tuple(final\_lst)  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 finally:  
 file\_name.close()  
  
  
def main():  
 line\_number("p1\_Diaz\_Jordan.py", "test\_file.txt")  
 print(parse\_functions("funs.py"))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal Session for problem 1 for funs.py

1 . # Jordan Diaz, This program manipulates files and parses itself  
2 . def line\_number(file\_name, write\_to\_this):  
3 . """ This file takes the contents of one file and copies it into another file but with numbered lines """  
4 .   
5 . # edge case  
6 . if file\_name == write\_to\_this:  
7 . print("Error: Cannot use the same file name")  
8 . exit(1)  
9 .   
10 . try:  
11 . # Open the files  
12 . file\_name = open(file\_name, "r")  
13 . write\_to\_this = open(write\_to\_this, "w")  
14 .   
15 . # Go through the file and write to the other  
16 . count = 1  
17 . for line in file\_name:  
18 . print(count, ". ", line, file=write\_to\_this, end="")  
19 . count += 1  
20 .   
21 . except FileNotFoundError as er:  
22 . print("The file: ", file\_name, " does not exist.")  
23 . print("Exception type: {} : the error message was {} ".format(type(er), er))  
24 . except PermissionError as er:  
25 . print("You do not have access to read the file: ", file\_name)  
26 . print("Exception type: {} : the error message was {} ".format(type(er), er))  
27 . finally:  
28 . # Close the files  
29 . file\_name.close()  
30 . write\_to\_this.close()  
31 .   
32 .   
33 . def parse\_functions(file\_name):  
34 . try:  
35 . file\_name = open(file\_name, "r")  
36 .   
37 . separation\_str = "def" + " "  
38 . curr\_function\_str = ""  
39 .   
40 . count = 1  
41 . lst\_of\_def\_names = []  
42 . lst\_of\_positions = []  
43 .   
44 . # Get the name of the function and the line number  
45 . for line in file\_name.readlines():  
46 .   
47 . for n in range(0, len(line)):  
48 . avoid\_str = "#" + " "  
49 . if n == line.find(avoid\_str) or n + 1 == line.find(avoid\_str):  
50 . break  
51 . elif n == line.find(" " + "#"):  
52 . break  
53 . elif line.find(separation\_str) != -1 and line[n - 1] == ":":  
54 . curr\_function\_str += "\n"  
55 . elif line.find("def") == 0:  
56 . curr\_function\_str += line[n]  
57 . elif line.find(" ") == 0:  
58 . curr\_function\_str += line[n]  
59 .   
60 . curr\_name = ""  
61 .   
62 . # Build a list of positions and names of functions  
63 . if line.find("def") == 0:  
64 . for i in range(4, line.find("(")):  
65 . curr\_name += line[i]  
66 . lst\_of\_positions.append(count)  
67 . lst\_of\_def\_names.append(curr\_name)  
68 .   
69 . count += 1  
70 .   
71 . separation\_str = "def" + " "  
72 .   
73 . lst\_of\_functions = curr\_function\_str.split(separation\_str)  
74 . lst\_of\_functions.pop(0)  
75 .   
76 . for k in range(0, len(lst\_of\_functions)):  
77 . lst\_of\_functions[k] = separation\_str + lst\_of\_functions[k].replace(" ", "\t")  
78 .   
79 . final\_lst = []  
80 . for m in range(0, len(lst\_of\_def\_names)):  
81 . tple = (lst\_of\_def\_names[m], lst\_of\_positions[m], lst\_of\_functions[m])  
82 . final\_lst.append(tple)  
83 .   
84 . final\_lst.sort()  
85 . return tuple(final\_lst)  
86 .   
87 . except FileNotFoundError as er:  
88 . print("The file: ", file\_name, " does not exist.")  
89 . print("Exception type: {} : the error message was {} ".format(type(er), er))  
90 . finally:  
91 . file\_name.close()  
92 .   
93 .   
94 . def main():  
95 . line\_number("p1\_Diaz\_Jordan.py", "test\_file.txt")  
96 . print(parse\_functions("funs.py"))  
97 .   
98 .   
99 . if \_\_name\_\_ == "\_\_main\_\_":  
100 . main()

Solution#2:

# Jordan Diaz, This program teaches a handful of python concepts  
def concatenate(separator\_char, \*strings):  
 *""" This program concatenates multiples strings separated by a specific character"""* temp\_str = ""  
 for i in range(0, len(strings)):  
 if i != len(strings) - 1:  
 temp\_str += strings[i] + separator\_char  
 elif i == len(strings) - 1:  
 temp\_str += strings[i]  
 return temp\_str  
  
  
# Pythagorean Triples  
pythagorean\_triples = [(i, j, k) for i in range(1, 100 + 1) for j in range(1, 100 + 1) for k in range(1, 100 + 1) if  
 ((i \*\* 2) + (j \*\* 2)) == (k \*\* 2)]  
  
print(pythagorean\_triples)  
  
# list of strings in a tuple  
lst = ["one", "seven", "three", "two", "tem"]  
  
lst\_of\_strings = [(len(elements), elements.capitalize()) for elements in lst if len(elements) > 3]  
  
print(lst\_of\_strings)  
  
# Reorder Names  
lst = ["Jules Verne", "Alexandre Dumas", "Maurice Druon"]  
  
final\_lst = ["{}, {}".format(new\_elements[1], new\_elements[0]) for new\_elements in  
 [elements.split() for elements in lst]]  
print(final\_lst)  
  
print(concatenate(": ", "one", "two", "three"))  
print(concatenate(" and ", "bonny", "Clyde"))  
print(concatenate(" and ", "single"))

Background pattern

Description automatically generated with medium confidenceTerminal Session for problem 2

Solution#3:

# Jordan Diaz, This program involves csv files  
import csv  
  
  
def add\_contact(contacts, info\_tuple):  
 *""" adds a contact to an already existing list of contacts"""* for contact in contacts:  
 if info\_tuple[0] == contact[0]:  
 contacts.remove(contact)  
 contacts.append(info\_tuple)  
 contacts.sort()  
 return False  
 contacts.append(info\_tuple)  
 contacts.sort()  
 return True  
  
  
def remove\_contact(contacts, info\_tuple):  
 *""" Removes a specific contact from a list of contacts"""* if info\_tuple in contacts:  
 contacts.remove(info\_tuple)  
 return True  
 return False  
  
  
def find\_contact(contacts, name=None, nickname=None):  
 *""" Finds a specific contact in a list of contact the name or nickname, must specify"""* for contact in contacts:  
 if name is not None:  
 if contact[0] == name:  
 return contact  
 elif nickname is not None:  
 if contact[1] == nickname:  
 return contact  
  
  
def save\_to\_csv(contacts, file\_name):  
 *""" This function changes the contents of a file to match a list of contacts"""* try:  
 with open(file\_name, "w", newline="") as file:  
 writer = csv.writer(file)  
  
 for contact in contacts:  
 writer.writerow([contact[0], contact[1], contact[2]])  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to open the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 finally:  
 file.close()  
  
  
def read\_from\_csv(file\_name):  
 *""" This function reads from a csv file and returns a list of tuples that has the contents of the file"""* try:  
 with open(file\_name, "r") as file:  
  
 # Read file and create a list of contacts, then returns that list  
 reader = csv.reader(file)  
 contacts\_in\_file = []  
 for line in reader:  
 contacts\_in\_file.append((line[0], line[1], line[2]))  
 return contacts\_in\_file  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to read the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
  
 finally:  
 file.close()  
  
  
def testif(b, test\_name, msg\_ok="", msg\_failed=""):  
 *""" This runs tests to reach the desired outcome of a function"""* if b:  
 print("Success: " + test\_name + "; " + msg\_ok)  
 else:  
 print("Failed: " + test\_name + "; " + msg\_failed)  
 return b  
  
  
def main():  
 *""" This is the main function, testing all the applicatons of the functions"""* # Test if you a contact is replaced and returns False  
 contact\_lst = []  
 add\_contact(contact\_lst, ("Earl Simmons", "DMX", "305-1010101"))  
 testif(add\_contact(contact\_lst, ("Earl Simmons", "DMX", "999-987654321")) == False and contact\_lst == [  
 ("Earl Simmons", "DMX", "999-987654321")], "Test 1")  
  
 # Test if a contact does not exist it will add it in alphabetical order and return true  
 testif(add\_contact(contact\_lst, ("Robert", "Rob", "123,3211233219")) == True and contact\_lst == [  
 ("Earl Simmons", "DMX", "999-987654321"), ("Robert", "Rob", "123,3211233219")], "Test 2")  
  
 # Test if a contact is removed it will return true  
 testif(remove\_contact(contact\_lst, ("Robert", "Rob", "123,3211233219")) == True and contact\_lst == [  
 ("Earl Simmons", "DMX", "999-987654321")], "Test 3")  
  
 # tests if a contact does not exist it will return False  
 testif(remove\_contact(contact\_lst, ("Robert", "Rob", "123,3211233219")) == False and contact\_lst == [  
 ("Earl Simmons", "DMX", "999-987654321")], "Test 4")  
  
 # tests if find contact returns the tuple using a name  
 testif(find\_contact(contact\_lst, name="Earl Simmons") == ("Earl Simmons", "DMX", "999-987654321"), "Test 5")  
  
 # tests if find contact returns the tuple using a nickname  
 testif(find\_contact(contact\_lst, nickname="DMX") == ("Earl Simmons", "DMX", "999-987654321"), "Test 6")  
  
 # Tests if cannot find contact it will return None  
 testif(find\_contact(contact\_lst, name="fred", nickname="freddy") is None, "Test 7")  
  
 # Tests if read and write to csv works  
 contact\_lst = []  
  
 add\_contact(contact\_lst, ("Earl Simmons", "DMX", "305-1010101"))  
 add\_contact(contact\_lst, ("Beyonce Knowles", "bey", "561-1234321"))  
 add\_contact(contact\_lst, ("Cardi B", "Belcalis", "305-4399521"))  
  
 save\_to\_csv(contact\_lst, "test.csv")  
  
 testif(  
 read\_from\_csv("test.csv") == [('Beyonce Knowles', 'bey', '561-1234321'), ('Cardi B', 'Belcalis', '305-4399521'),  
 ('Earl Simmons', 'DMX', '305-1010101')], "Test 8")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal Session for problem 3:

Text

Description automatically generated

Test.csv file

Text

Description automatically generated

Solution#4

# Jordan Diaz, This program manages rankings from csv files  
import csv  
  
  
def read\_from\_top\_casts(file\_name, director\_movies\_dict, actor\_one\_movies\_dict):  
 *""" Reads from the top casts file and modifies two dictionaries based on a csv file"""* try:  
 with open(file\_name, "r", encoding="utf-8") as file:  
  
 reader = csv.reader(file)  
 lst\_of\_directors\_and\_actor1 = []  
  
 # Go line by line creating a dictionary of the movie[director] and movie[actor]  
 for line in reader:  
 title = line[0]  
 director = line[2]  
 actor\_one = line[3]  
  
 # Check if sets exist  
 if not (director in director\_movies\_dict):  
 director\_movies\_dict[director] = set()  
 if not (actor\_one in actor\_one\_movies\_dict):  
 actor\_one\_movies\_dict[actor\_one] = set()  
  
 # Add to set and list  
 director\_movies\_dict[director].add(title)  
 actor\_one\_movies\_dict[actor\_one].add(title)  
 lst\_of\_directors\_and\_actor1.append((director, actor\_one))  
  
 lst\_of\_tuples = []  
  
 # Build the list of tuples and returns it  
 for items in lst\_of\_directors\_and\_actor1:  
 the\_director = items[0]  
 the\_actor = items[1]  
  
 # uses set theory to find the intersection of the two sets  
 shared\_movies\_count = len(director\_movies\_dict[the\_director] & actor\_one\_movies\_dict[the\_actor])  
 lst\_of\_tuples.append((the\_director, the\_actor, shared\_movies\_count))  
  
 return lst\_of\_tuples  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to read the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
  
 finally:  
 file.close()  
  
  
def read\_from\_top\_rated(file\_name):  
 *"""Reads from the top rated file and returns a list of all the movies"""* try:  
 with open(file\_name, "r", encoding="utf-8") as file:  
  
 reader = csv.reader(file)  
 lst\_of\_top\_rated = []  
  
 for line in reader:  
 title = line[1]  
 lst\_of\_top\_rated.append(title)  
  
 lst\_of\_top\_rated.pop(0)  
 return lst\_of\_top\_rated  
  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to read the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
  
 finally:  
 file.close()  
  
  
def read\_from\_top\_grossing(file\_name, box\_office\_dict):  
 *""" Reads from top grossing and alters a dictionary to match the contents of the title and box office of each  
 rank """* try:  
 with open(file\_name, "r", encoding="utf-8") as file:  
  
 reader = csv.reader(file)  
 for line in reader:  
 title = line[1]  
 box\_office = line[3]  
  
 if title != "Title":  
 if not (title in box\_office\_dict):  
 box\_office\_dict[title] = set()  
 box\_office\_dict[title].add(box\_office)  
  
 return box\_office\_dict  
  
 except FileNotFoundError as er:  
 print("The file: ", file\_name, " does not exist.")  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
 except PermissionError as er:  
 print("You do not have access to read the file: ", file\_name)  
 print("Exception type: {} : the error message was {} ".format(type(er), er))  
  
 finally:  
 file.close()  
  
  
def display\_top\_collaborations(display=None):  
 *""" Displays ranking of a tuple based on if specific elements of the tuple are in another file"""* movie\_directors = dict()  
 movie\_actors1 = dict()  
  
 # Get the data from the functions  
 lst\_of\_tuples = read\_from\_top\_casts("imdb-top-casts.csv", movie\_directors, movie\_actors1)  
 lst\_of\_movies\_in\_top\_rated = read\_from\_top\_rated("imdb-top-rated.csv")  
 final\_set\_of\_tuples = set()  
  
 # Filter what is in the top rated list and top cast  
 for i in range(0, len(lst\_of\_tuples)):  
 for d\_movies in movie\_directors[lst\_of\_tuples[i][0]]:  
 if d\_movies in lst\_of\_movies\_in\_top\_rated:  
 final\_set\_of\_tuples.add(lst\_of\_tuples[i])  
 for a\_movies in movie\_actors1[lst\_of\_tuples[i][1]]:  
 if a\_movies in lst\_of\_movies\_in\_top\_rated:  
 final\_set\_of\_tuples.add(lst\_of\_tuples[i])  
  
 # Sort and Display  
 count = 1  
 final\_lst = list(final\_set\_of\_tuples)  
 final\_lst.sort(key=lambda x: x[2], reverse=True)  
 if display is None:  
 for element in final\_lst:  
 print("#{}. ".format(count), element)  
 count += 1  
 elif 1 < display < len(final\_lst):  
 for index in range(0, display):  
 print("#{}. ".format(count), final\_lst[index])  
 count += 1  
  
  
def display\_top\_directors(display=None):  
 *""" displays the ranking of movie directors from the top grossing list ordered by the total box office money they  
 produced """* # prepare dictionaries and get data from functions  
 dict\_of\_director = dict()  
 dict\_of\_actor = dict()  
 dict\_of\_top\_grossing = dict()  
 dict\_of\_top\_grossing = read\_from\_top\_grossing("imdb-top-grossing.csv", dict\_of\_top\_grossing)  
 read\_from\_top\_casts("imdb-top-casts.csv", dict\_of\_director, dict\_of\_actor)  
  
 # Filter out what can and cannot be use  
 final\_lst = []  
 for items in dict\_of\_director:  
 director = items  
 movies\_lst = dict\_of\_director[director]  
 for movie in movies\_lst:  
 if movie in dict\_of\_top\_grossing:  
 final\_lst.append((director, dict\_of\_top\_grossing[movie].pop(), movie))  
  
 # Sort and Display  
 count = 1  
 final\_lst.sort(key=lambda x: x[1], reverse=True)  
 if display is None:  
 for element in final\_lst:  
 print("#{}. ".format(count), element)  
 count += 1  
 elif 1 < display < len(final\_lst):  
 for index in range(0, display):  
 print("#{}. ".format(count), final\_lst[index])  
 count += 1  
  
  
def main():  
 *""" This is the main function used for testing"""* display\_top\_collaborations(display=5)  
 print()  
 display\_top\_directors(display=5)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Terminal Session for problem 4

Text

Description automatically generated