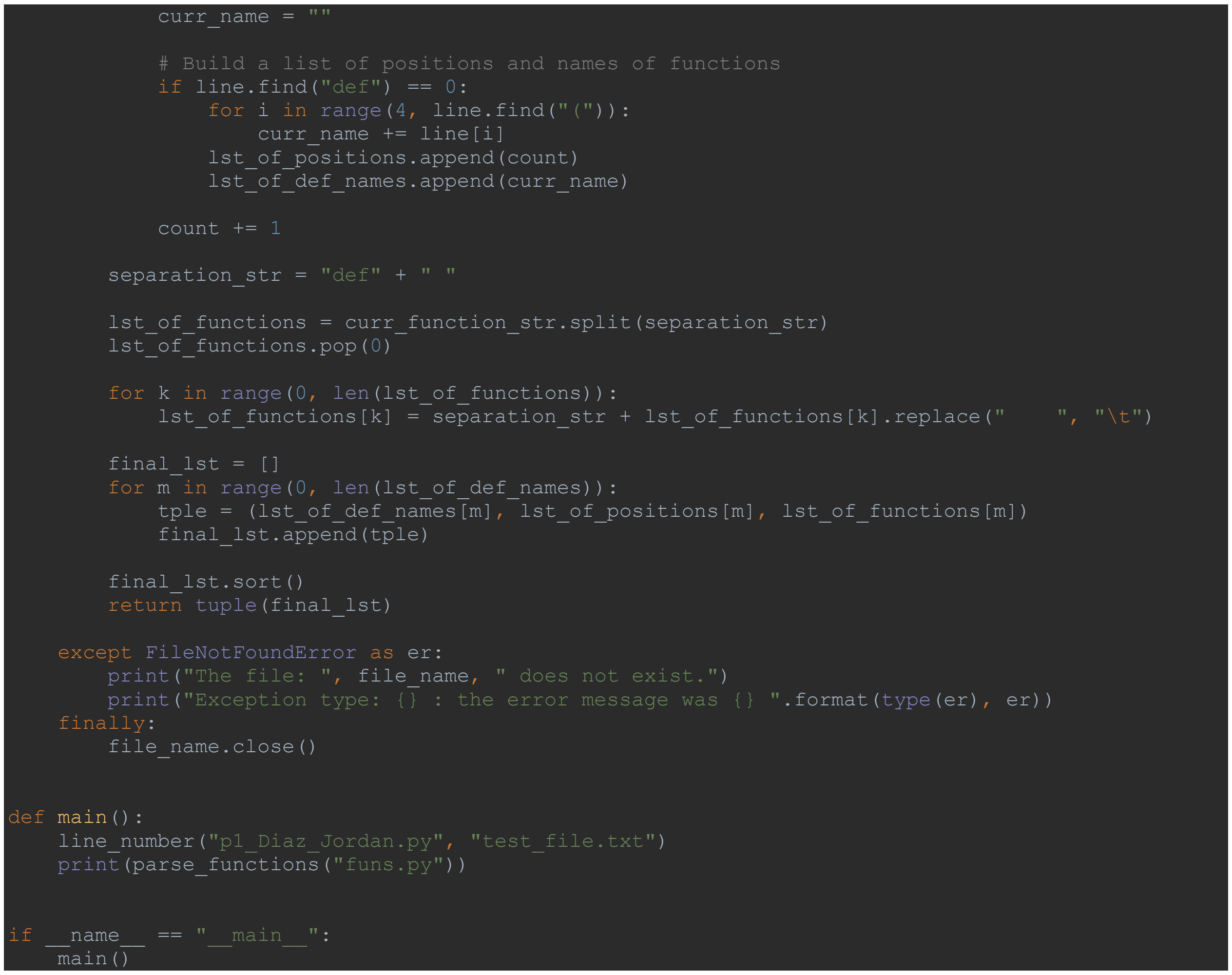
Python Programming

Homework 2

Jordan Diaz

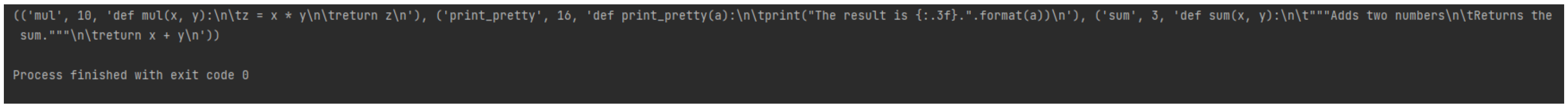
Solution #1:

```python
# 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]
```



```python
            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



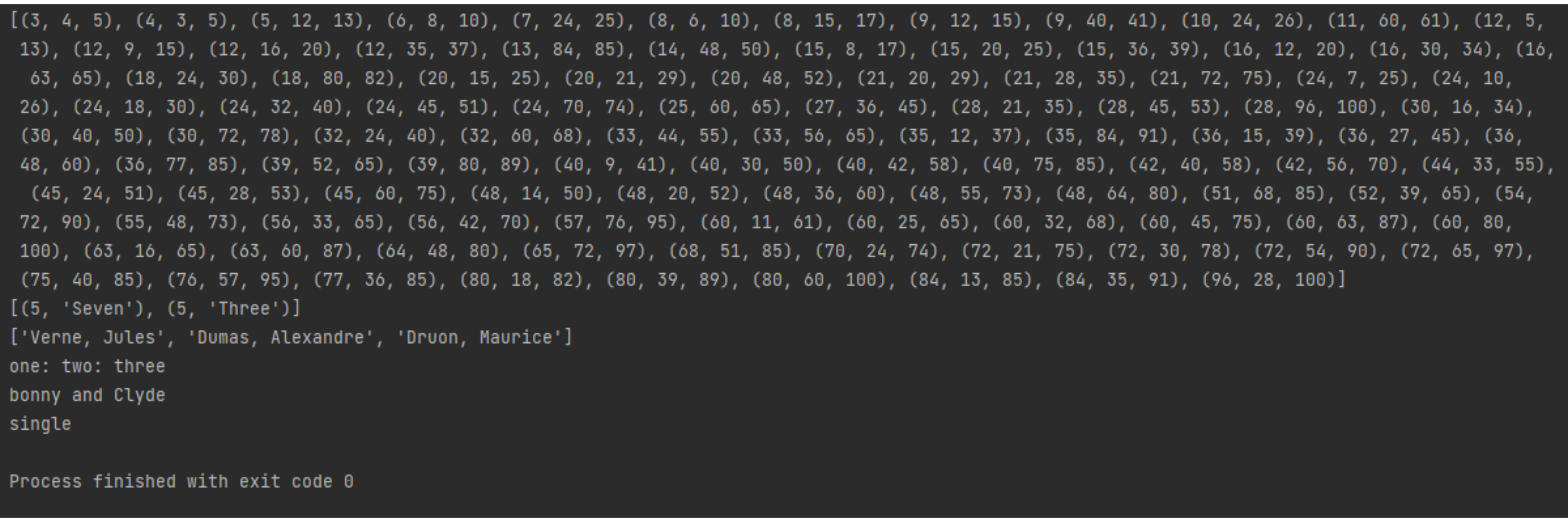
```
(('mul', 10, 'def mul(x, y):\n\tz = x * y\n\treturn z\n'), ('print_pretty', 16, 'def print_pretty(a):\n\tprint("The result is {:.3f}.".format(a))\n'), ('sum', 3, 'def sum(x, y):\n\t"""Adds two numbers\n\tReturns the
 sum."""\n\treturn x + y\n'))

Process finished with exit code 0
```

```
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] ==
":":
                    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()
```

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"))
```

Terminal Session for problem 2



```
[(3, 4, 5), (4, 3, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 6, 10), (8, 15, 17), (9, 12, 15), (9, 40, 41), (10, 24, 26), (11, 60, 61), (12, 5,
 13), (12, 9, 15), (12, 16, 20), (12, 35, 37), (13, 84, 85), (14, 48, 50), (15, 8, 17), (15, 20, 25), (15, 36, 39), (16, 12, 20), (16, 30, 34), (16,
  63, 65), (18, 24, 30), (18, 80, 82), (20, 15, 25), (20, 21, 29), (20, 48, 52), (21, 20, 29), (21, 28, 35), (21, 72, 75), (24, 7, 25), (24, 10,
 26), (24, 18, 30), (24, 32, 40), (24, 45, 51), (24, 70, 74), (25, 60, 65), (27, 36, 45), (28, 21, 35), (28, 45, 53), (28, 96, 100), (30, 16, 34),
 (30, 40, 50), (30, 72, 78), (32, 24, 40), (32, 60, 68), (33, 44, 55), (33, 56, 65), (35, 12, 37), (35, 84, 91), (36, 15, 39), (36, 27, 45), (36,
 48, 60), (36, 77, 85), (39, 52, 65), (39, 80, 89), (40, 9, 41), (40, 30, 50), (40, 42, 58), (40, 75, 85), (42, 40, 58), (42, 56, 70), (44, 33, 55),
  (45, 24, 51), (45, 28, 53), (45, 60, 75), (48, 14, 50), (48, 20, 52), (48, 36, 60), (48, 55, 73), (48, 64, 80), (51, 68, 85), (52, 39, 65), (54,
 72, 90), (55, 48, 73), (56, 33, 65), (56, 42, 70), (57, 76, 95), (60, 11, 61), (60, 25, 65), (60, 32, 68), (60, 45, 75), (60, 63, 87), (60, 80,
 100), (63, 16, 65), (63, 60, 87), (64, 48, 80), (65, 72, 97), (68, 51, 85), (70, 24, 74), (72, 21, 75), (72, 30, 78), (72, 54, 90), (72, 65, 97),
 (75, 40, 85), (76, 57, 95), (77, 36, 85), (80, 18, 82), (80, 39, 89), (80, 60, 100), (84, 13, 85), (84, 35, 91), (96, 28, 100)]
[(5, 'Seven'), (5, 'Three')]
['Verne, Jules', 'Dumas, Alexandre', 'Druon, Maurice']
one: two: three
bonny and Clyde
single

Process finished with exit code 0
```

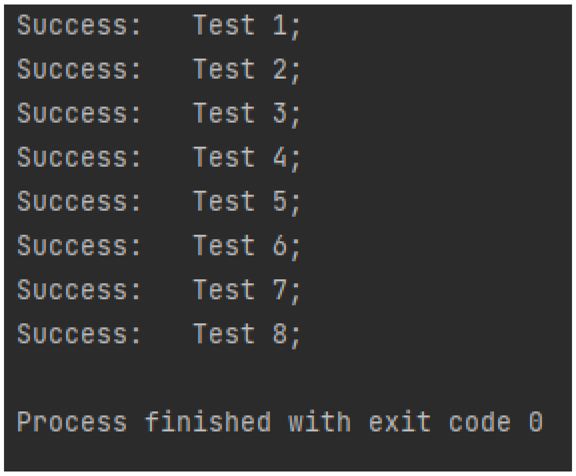
Solution#3:

```python
# 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
```

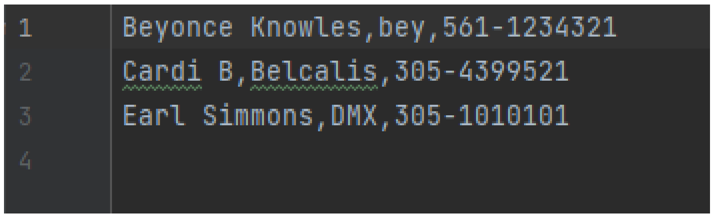
```python
    testif(remove_contact(contact_lst, ("Robert", "Rob", "123,3211233219"))
== True and contact_lst == [
        ("Earl Simmons", "DMX", "999-987654321")], "Test 3")

    # 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:



```
Success:   Test 1;
Success:   Test 2;
Success:   Test 3;
Success:   Test 4;
Success:   Test 5;
Success:   Test 6;
Success:   Test 7;
Success:   Test 8;

Process finished with exit code 0
```

Test.csv file



```
Beyonce Knowles,bey,561-1234321
Cardi B,Belcalis,305-4399521
Earl Simmons,DMX,305-1010101

```

Solution#4

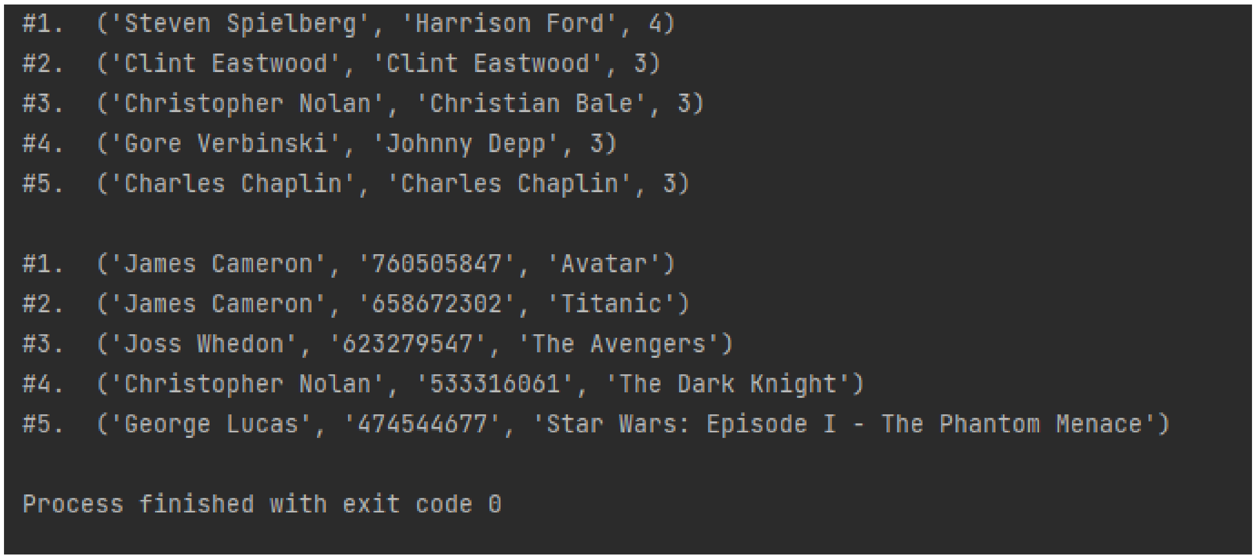
```
# Jordan Diaz, This program manages rankings from csv files
import csv


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)
```

```python
    # 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



```
#1.  ('Steven Spielberg', 'Harrison Ford', 4)
#2.  ('Clint Eastwood', 'Clint Eastwood', 3)
#3.  ('Christopher Nolan', 'Christian Bale', 3)
#4.  ('Gore Verbinski', 'Johnny Depp', 3)
#5.  ('Charles Chaplin', 'Charles Chaplin', 3)

#1.  ('James Cameron', '760505847', 'Avatar')
#2.  ('James Cameron', '658672302', 'Titanic')
#3.  ('Joss Whedon', '623279547', 'The Avengers')
#4.  ('Christopher Nolan', '533316061', 'The Dark Knight')
#5.  ('George Lucas', '474544677', 'Star Wars: Episode I - The Phantom Menace')

Process finished with exit code 0
```