```python
import os
import json
import unittest
__author__ = 'Avital & Chen'
# Avital Kahani 205688666
# Chen Gutman 205616147


# test that check all the functions
class TestAddressBook(unittest.TestCase):
   def setUp(self):
       folder_name = "testing_address_book"
       self.address_book = AddressBook(folder_name)

   def tearDown(self):
       for filename in os.listdir('testing_address_book'):
           os.remove(os.path.join("testing_address_book",filename))
       os.rmdir("testing_address_book")

   def test_init_address_book(self):
       expected_folder_name = "testing_address_book"
       for filename in os.listdir('.'):
           if os.path.isdir(filename):
               if expected_folder_name == filename:
                   actual_folder_name = filename
       self.assertEqual(expected_folder_name, actual_folder_name,
"faild test for init")

   def test_add_contact(self):
       contact = Contact("first", "last", "050121212")
       expected_content = '{"lastname": "last", "telephone":
"050121212", "firstname": "first"}'
       self.address_book.add_contact(contact)
       with open(os.path.join("testing_address_book",
"1_first_last"), 'r') as f:
           actual_content = f.read()
           self.assertEqual(expected_content, actual_content, "faild
test for add_contant")

   def test_get_contact(self):
       contact = Contact("first", "last", "050121212")
       self.address_book.add_contact(contact)
       actual_contact_list = self.address_book.get_contacts()
       expected_values = ["first", "last", "050121212"]
       actual_values = [actual_contact_list[0].firstname,
actual_contact_list[0].lastname, actual_contact_list[0].telephone]
       self.assertEqual(expected_values, actual_values, "failed test
for get_contact")
```

```
    def test_find_contact(self):
        contact = Contact("first", "last", "050121212")
        self.address_book.add_contact(contact)
        expected_values = ["first", "last", "050121212"]
        actual_contact_found = self.address_book.find_contact("first",
"last", "050121212")
        actual_values = [actual_contact_found[0].firstname,
actual_contact_found[0].lastname, actual_contact_found[0].telephone]
        self.assertEqual(expected_values, actual_values, "failed test
for find_contact")


class AddressBookException(Exception):
    pass


class AddressBook:
    # constructor to create a folder
    def __init__(self, folder_name):
        self.folder_name = folder_name
        self.contact_counter = 1
        try:
            os.mkdir(folder_name + '/')
        except:
            raise AddressBookException("Cannot Create " + folder_name
+ ", address book with this name already exists")

    # function that create new file for a contact and add him to
address book folder.
    def add_contact(self, contact):
        file_name = str(self.contact_counter) + '_' +
contact.firstname + '_' + contact.lastname
        file_path = os.path.join(self.folder_name, file_name)
        new_contact = open(file_path, "w+")
        new_contact.write(contact.__str__())

    # function that return the list of all the contacts in the address
book folder
    def get_contacts(self):
        contact_list = list()
        for filename in os.listdir(self.folder_name):
            with open(os.path.join(self.folder_name, filename), 'r')
as f:
                contact_json = json.load(f)
                contact = Contact(contact_json["firstname"],
contact_json["lastname"], contact_json["telephone"])
                contact_list.append(contact)
```

```python
        return contact_list

    # function that search for a specific contact in the address book
folder
    def find_contact(self, firstname, lastname, telephone):
        contact_list = self.get_contacts()
        found_contact = list()
        for contact in contact_list:
            if firstname == contact.firstname and lastname ==
contact.lastname and telephone == contact.telephone:
                found_contact.append(contact)
        return found_contact


class Contact:
    # constructor to create a contact object
    def __init__(self, firstname, lastname, telephone):
        self.firstname = firstname
        self.lastname = lastname
        self.telephone = telephone

    # function that convert the object to string
    def __str__(self):
        contact_json = {
            "firstname": self.firstname,
            "lastname": self.lastname,
            "telephone": self.telephone
        }
        return json.dumps(contact_json)


def main():

    contact1 = Contact("David", "Cohen", "054617876")
    contact2 = Contact("Sholamit", "Levi", "054617656")
    contact3 = Contact("Ben", "Benjamin", "054617123")

    try:
        new_address_book = AddressBook('my_address_book')
        new_address_book.add_contact(contact1)
        new_address_book.add_contact(contact2)
        new_address_book.add_contact(contact3)

        address_book_contact_list = new_address_book.get_contacts()

        print("Contact List:")
        for contact in address_book_contact_list:
            print(contact)
        found_contacts = new_address_book.find_contact("Sholamit",
"Levi", "054617656")
        print("Found Contacts")
        for found_contact in found_contacts:
            print(found_contact)

    except AddressBookException as exception_string:
        print(exception_string)

    # Showing Exeption for creating an address book with an existing
name
    try:
        exep_address_book = AddressBook('my_address_book')
    except AddressBookException as exception_string:
        print(exception_string)

if __name__ == "__main__":
    main()
```