Alternate Project 7: Mailing List Created by Brian Weinfeld In this project, you will use dictionaries, lists and objects with Python to create a program that generates mailing lists for advertisers. ## Overview Every company stores data on its users and a common use of this data is to send relevant advertisements. If the company can track your interests, they can email you when they are selling something they think you may want to purchase. This project consists of several common functions designed to identify which customers you want to send advertisements to. ## Details ### Behavior ```python What would you like to do? (add, count, send, display, exit) add Enter name email and hobbies: Alice a_person@host.com horses,traveling a_person@host.com is not in our list a_person@host.com has been added to our list What would you like to do? (add, count, send, display, exit) add Enter name email and hobbies: Blake myemail@host.com school myemail@host.com is not in our list myemail@host.com has been added to our list What would you like to do? (add, count, send, display, exit) display a_person@host.com Alice ['horses', 'traveling'] myemail@host.com Blake ['school'] What would you like to do? (add, count, send, display, exit) add Enter name email and hobbies: Alice a_person@host.com sleeping a_person@host.com is already in our list Added sleeping to a_person@host.com' hobbies What would you like to do? (add, count, send, display, exit) display a_person@host.com Alice ['horses', 'traveling', 'sleeping'] myemail@host.com Blake ['school'] What would you like to do? (add, count, send, display, exit) count Hobby Results {'horses': 1, 'traveling': 1, 'sleeping': 1, 'school': 1} What would you like to do? (add, count, send, display, exit) send Which hobby? horses Mailing horses to: ['a_person@host.com'] What would you like to do? (add, count, send, display, exit) exit Goodbye ``` ### Implementation Details * Begin by completing the Person class. The person class represents each person your company is tracking. __name__ represents the person's name, __email__ represents their email, and __hobbies__ is a list of strings representing the things you know this person is interseted in. The list may be empty but there should never be repeats (you don't want the same hobby for the same person listed twice). While it is possible for many people to have the same __name__ and __hobbies__, we will assume that each __email__ is unique. That is, only one person can have a specific email address. ```python class Person: def __init__(self, name, email, hobbies): # finish def __str__(self): # finish ``` * Next, complete the 4 methods in the Mailer class. These will be used to create and interact with a list of people. They are described in more detail below. ```python class Mailer: def __init__(self): self.people = [] def __str__(self): return '\n'.join(str(p) for p in self.people) def send_hobby_mailer(self, hobby): # finish def count_hobbies(self): # finish def already_present(self, check): # finish def add_person(self, check): # finish ``` * In __send_hobby_mailer__ create a list of emails that contain all the people who you know are interested in that hobby. Print the list. * In __count_hobbies__ print each hobby and the number of people who have it as a hobby. This will let you determine which products your company should carry. * __already_present__ has one parameter check that is of type Person. This function determines whether this person is already in our mailing list. If they are not in our mailing list, print a message and return None. If they are in our mailing list, print a message and return the person. * __add_person__ has one parameter check that is of type Person. This function does one of two actions. If the person is not in the mailing list, add them to the mailing list. If the person is already in the mailing list, we don't want to add them again. Instead, add the hobbies listed to their already identified hobbies to create a new, possibly longer list of hobbies. Be careful to not add any hobbies that are already in the list. Below if an example of the functionality of the four functions. ```python > mailer = Mailer() > mailer.add_person(Person('Alice', 'a_dog@host.com', ['dogs', 'animals'])) a_dog@host.com is not in our list a_dog@host.com has been added to our list > mailer.add_person(Person('Bob', 'knitting@host.com', ['knitting', 'surfing', 'painting'])) knitting@host.com is not in our list knitting@host.com has been added to our list > mailer.add_person(Person('Carlos', 'filmbuff@host.com', ['movies'])) filmbuff@host.com is not in our list filmbuff@host.com has been added to our list > mailer.add_person(Person('Daisy', 'soccerfan@host.com', ['soccer', 'tennis', 'dogs'])) soccerfan@host.com is not in our list soccerfan@host.com has been added to our list > mailer.add_person(Person('Eva', 'everything@host.com', ['surfing', 'movies', 'knitting', 'animals'])) everything@host.com is not in our list everything@host.com has been added to our list > print(mailer) a_dog@host.com Alice ['dogs', 'animals'] knitting@host.com Bob ['knitting', 'surfing', 'painting'] filmbuff@host.com Carlos ['movies'] soccerfan@host.com Daisy ['soccer', 'tennis', 'dogs'] everything@host.com Eva ['surfing', 'movies', 'knitting', 'animals'] > mailer.add_person(Person('Bob', 'another_bob@host.com', ['cats', 'tennis'])) another_bob@host.com is not in our list another_bob@host.com has been added to our list > mailer.add_person(Person('Bob', 'knitting@host.com', ['surfing', 'animals', 'poetry'])) knitting@host.com is already in our list Added animals to knitting@host.com' hobbies Added poetry to knitting@host.com' hobbies > print(mailer) a_dog@host.com Alice ['dogs', 'animals'] knitting@host.com Bob ['knitting', 'surfing', 'painting', 'animals', 'poetry'] filmbuff@host.com Carlos ['movies'] soccerfan@host.com Daisy ['soccer', 'tennis', 'dogs']

everything@host.com Eva ['surfing', 'movies', 'knitting', 'animals'] another_bob@host.com Bob ['cats', 'tennis'] > mailer.count_hobbies() Hobby Results {'dogs': 2, 'animals': 3, 'knitting': 2, 'surfing': 2, 'painting': 1, 'poetry': 1, 'movies': 2, 'soccer': 1, 'tennis': 2, 'cats': 1} > mailer.send_hobby_mailer('animals') Mailing animals to: ['a_dog@host.com', 'knitting@host.com', 'everything@host.com'] > mailer.send_hobby_mailer('tennis') Mailing tennis to: ['soccerfan@host.com', 'another_bob@host.com'] > mailer.send_hobby_mailer('unique') Mailing unique to: [] ``` * Finally, create a loop that allows a user to enter commands __add__, __count__, __send__, __display__, __exit__ to interact with the code you have already created. ### Challenge This section contains additional components you can add to the project. These should only be attemped after the project has been completed. * As in previous projects, change the looping code so that all typed information can be done at once, instead of spread over multiple lines * Create a new function __send_hobby_mailer_any__. The hobby parameter in this function is a list of hobbies. Create a list of emails that includes a person if they enjoy ANY of the hobbies listed. Modify the loop to allow this option to be selected. * Create a new function __send_hobby_mailer_all__. The hobby parameter in this function is a list of hobbies. Create a list of emails that includes a person if they enjoy ALL of the hobbies listed. Modify the loop to allow this option to be selected. ### Super Challenge The super challenge will require knowledge that has not been taught yet. You will need to do additional research on your own. Good luck! A common problem that data engineers face is unclean data. That is, data that is not perfectly entered by the user. For example, capitialization does not matter in an email address. My_email@host.com and my_email@host.com will go to the same place. Punctuation like periods also does not matter. MyEmail@host.com and My.Email@host.com will go to the same place. Right now, your code cannot tell the difference between these two and will treat them as different people. Part of a data engineer's job is to clean all data inputs. Clean all of the emails in your program as they are entered. Standardize how they are stored so that you can now tell if unclean data entrees like those above are actually the same address. What other types of mistakes might a user make? Try to fix those as well!