CSCA48 Assignment 1

Summer 2018

Due: June 23rd 2018, 5 pm

Description:

In this assignment you will practice developing an ADT by implementing a phonebook. Suppose that you are working in a team of software engineers and your job is to design and implement the required code for this phonebook while your teammate takes care of the graphical user interface. This means that in the final product, by touching an icon, one of the methods that you've implemented will be called.

A phonebook is a repository in which you can store people's cell phone number, first name, surname, home phone, work phone, email and their home address. For a person to be inserted in this repository, at least one phone number is required, which can be either cell phone, home or work phone. The first phone number available (not None) amongst these three is called *primary* phone number. Also, at least one contact name (i.e. either the first name or surname) should be provided. In this description, whenever a contact's name is referred to, we actually mean contact surname. In case that the surname is not provided for a contact, contact's name refers to the contact's first name. All other attributes of a contact are optional and are not required for storing the contact in the phonebook. We assume that two contacts are the same if a) the provided name is the same in terms of both the type (i.e. first name and surname) and the value and b) if the provided primary phone number is from the same type (e.g. both primary phone numbers are cell phone number), therefore the number itself is the same.

This phonebook should also allow you to label some of the contacts as your favourite contacts. The phonebook itself is a single linked list in which you store objects that represent the contacts.

The operations that are available by this phonebook include:

1- `add(contact)`: add the given contact in the right place in alphabetical order. If both the first name and surname is provided for the contact, then *surname* is the basis for inserting the contact in the right place in alphabetical order, otherwise *first name* is used. When inserting the contacts in their right places, characters cases should not matter. e.g. "John" is the same as "jOhN"

2- `remove(contact)`: for the given contact, you should be able to remove it from your phonebook and return the contact. If contact was not found, None should be returned.

3- `show_fav_list()`: Returns a list of all the names that are labeled as your favourite contact. If both the surname and first name for a contact had been included only the surname, otherwise the first name, is returned.

4- `toronto_phone()`: Toronto phone numbers start with either 416 or 647. This method returns a tuple containing two phonebooks, one includes the contacts whose *primary* phone number starts with 416 and the seconds starts with 647.

5- `reversed_phonebook()`: while phonebooks normally store the contacts in ascending order (i.e. a to z), it should be possible to see the list of the contacts in descending order. This method returns a phonebook containing the contacts that are listed in descending order (z to a).

6- `sync_phonebook(phonebook)`: when customers buy a new device, they would like to move their phonebook from their old device to new one. This method merges the current phonebook with the given phonebook that is given as an input to this method. When these two phonebooks are merged, their alphabetical order should still be preserved. In case two contacts are the same, all their information should be merged and added as one contact. Two contacts are the same if __eq__ method returns true (see *extra implementation information* for this).

7- `group_remove(remove_list)`: It is possible that you want to remove a group of contacts, which is given in a form of a list, from your phonebook. This method does this job for you.

8- `get_sublist(letter)`: This method returns a phonebook where the contacts' name starts with the given letter.

9- `get_housemate  (home_phone)`: This method returns a phonebook containing all the contacts, who live in the same house and share the same home phone number.

10- `rearrange_phonebook()`: Since your favorite contacts are the one that are accessed more frequently, this method rearranges the phonebook in a way that all the favourite contacts are in the front of the phonebook. After rearranging the phonebook, both the favourite and non-favourite contacts should be in an alphabetical order.

Extra implementation information:

- Cell phone numbers are unique but home phone and work phone can be shared amongst a group of people.
- It is possible that two contacts have the same first name and surname.
- Name of the methods should be exactly the same as what is mentioned in this description.
- All the attributes in both Contact and PhoneBook are private variables and should not be accessed directly.
- No *print* statements and any other *import* statements other than what is inside the starter code should exist in your code.
- Feel free to write any helper method that you require!
- Make sure you implement __eq__ in `Contact` class by which you can compare two contacts for equality.
- Do not modify the starter code.

How to best approach solving this problem:

- For each method write a few test cases to test the correctness of the method. Writing adversarial test cases is not required here. As we assume that there is an external module that tests all the inputs before it sends it to the methods that you've written.
- Hopefully in this stage, you are clear about what each method does. Now you can implement the methods for which you need to download PhoneBook.py. Don't forget to fill in the declaration of the originality of the code.
- Test your method against the test cases that you have designed in first step. If there exists any error, correct your code and run the test cases again.