Python
Contacts Application

Gaurav Gupta
tuteur.py@gmail.com

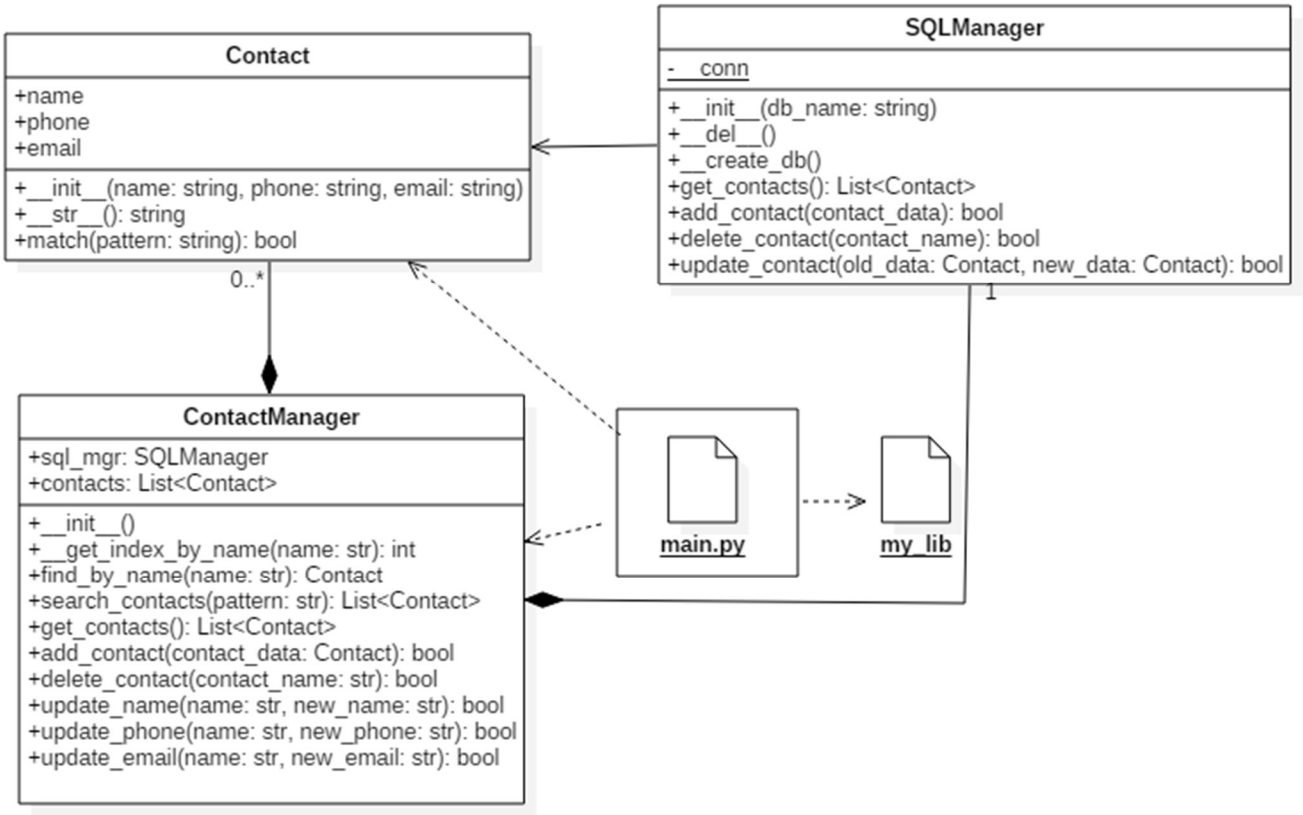
Convention:

- **Use functions from my_lib when possible.**
- **Add Doc Strings for all Classes, Modules and Functions**

Prerequisite:

- **Classes, Operator Overloading, Sqlite3 in Python**
- **UML Class Diagrams**

Contacts application keeps track of Contact data: Name, Phone No, Email.
All the data is maintained in form of SQLite Database(DB).
The App provides multiple options to view, modify and delete contacts.



Refer to the above class diagram and implement the Classes and Scripts as described below.

Gaurav Gupta
tuteur.py@gmail.com

Class - Contact

Script - contacts.py

Attributes:

1. **name <str>:** instance Attribute to store name
2. **phone <str>:** instance Attribute to store phone
3. **email <str>:** instance Attribute to store email

Methods:

1. **Constructor**
   initialize the 3 data members: name, phone, email. Email takes a default value of empty string.
2. **string conversion**
   add a string conversion operator overload, to return a single line formatted string
3. **match()**
   match function takes a string pattern to match the name with. Matches should be case insensitive and partial matches should be supported.

Gaurav Gupta
tuteur.py@gmail.com

Class - SQLHandler Script - sql_handler.py

Attributes:

1. **__conn** : class Attribute to store active connection

Methods:

1. **Constructor**
   Single argument, to initialize the name of database file. Initialize **__conn** with a connection to sqlite DB with the name as the database file. Call the **__create_db()** method.
2. **Destructor**
   close the connection (**__conn**) and set it to **None**.
3. **string conversion**
   add a string conversion operator overload (**__str__**), to return a single line formatted string
4. **cleanup_db()**
   delete all data from the database.
5. **__create_db()**
   execute the Create Query for the Contact table.
6. **get_contacts()**
   query the Contact table for all data and return a **list of Contact Objects**.
7. **add_contact(contact_data)**
   takes a Contact object as argument and executes insert query. Returns True if insert succeeds False otherwise. Add exception handling, to handle SQLite Errors.
8. **delete_contact(contact_name)**
   takes a **contact name** as argument and executes a delete query on the table with name as the matching criterion. Returns status of the delete as True or False.
9. **update_contact(old_data, new_data)**
   takes 2 Contact objects as arguments and replaces the details of old_data with all the details of new_data in the DB using update Query. Returns the status of Update.

Gaurav Gupta
tuteur.py@gmail.com

Class - ContactManager Script - contact_manager.py

Attributes:

1. **sql_mgr <SQLHandler>:** instance of SQLHandler
2. **contacts <list>:** instance Attribute to store list of Contacts

Methods:

1. **Constructor**
   No argument constructor to create and object of SQLHandler with Database name as 'contact.db'. Invoke the *get_contacts* method of SQLHandler object ot initialize the **contacts** list.
2. **__get_index_by_name(name)**
   Takes a name argument and returns the index of matching contact object from the *contacts*. Returns -1 if not found in list.
3. **find_by_name(name)**
   Call the __get_index_by_name method and get the corresponding matching index. Based in the index return the corresponding Contact object from *contacts* list, other wise return *None*.
4. **search_contacts(pattern)**
   return a list of all contacts for which the contact's name matches the pattern.
   i.e. return all Contact objects for which the *match_name* method returns True.
5. **get_contacts()**
   return all contacts list.
6. **add_contact(contact_data)**
   Takes a Contact object. Invokes SQLHander to add new contact.
   Adds to the ***contacts*** list if SQLHandler ***add_contact*** method return True.
   Returns final status as True or False.
7. **delete_contact(contact_name)**
   Takes a contact_name string as argument.
   Calls delete on SQLHandler.
   On success from SQLHandler, searches for a contact with that name in the list of contacts (__get_index_by_name) and deletes it from the list.
   Returns final status as bool.
8. **update_name(name, new_name)**
   name: contact to be updated
   new_name: name to be updated
   Search for the contact with **name** using __get_index_by_name.
   If found in search, create a copy of the corresponding contact object.

Update the name in new object and invoke SQLManager's *update_contact* method with old and new data.
If update_contact returns success, update the local copy and return the status.

9. **update_phone(name, new_phone)**
   name: contact to be updated
   new_phone: phone field to be updated
   Implementation similar to **update_name()**.
10. **update_email(name, new_email)**
    name: contact to be updated
    new_email: email field to be updated
    Implementation similar to **update_name()**.

Gaurav Gupta
tuteur.py@gmail.com

Script - main.py

This is the Main Script to start the application.

Following options are displayed to the user

1. **View**
2. **Add**
3. **Delete**
4. **Update**
5. **Exit**

Based on user Input, various operations are performed using the ContactManager class.

1. **View option further provides following options**

   View Contacts

   a. **Enter Name**
   b. **Search**
   c. **View All**
2. **Add option asks for Contact data and adds new contact.**
3. **Delete option asks for Contact name to be deleted and deletes from DB**
4. **Update Option has following sub options:**

   Update Contacts
   a. **Update Name**
   b. **Update Phone**
   c. **Update Email**
5. **Exit from the application.**