(https://colab.research.google.com/github/awash-analytics/mastawesha-address-book-app/blob/master/blog_addressBookApp_20181209.ipynb)

Mastawesha: Address Book Application built using Python language

Mastawesha means an address book in Amharic. Awash Analytics (http://awash-analytics.info/) recently built address book application using a [Python](https://en.wikipedia.org/wiki/Python_(programming_language%29) (https://en.wikipedia.org/wiki/Python_(programming_language%29)) programming language. The address book project is described here (http://awash-analytics.info/project-detail/?pdb=25), and it belongs to Martyr2 programming challenges which was posted on our website sometimes ago. In this blog post, I walk you through the source code used by the AddressBook App, and with a demo showing you how to signup as a new user of the app and then how to login to the app. In the demo, I show you how to add friends into your contacts list. Let's dive into it.

The User Interface: A Gateway to AddressBook App

The first time you run the AddressBook app, it displays you an interface with three options: (1) Login, (2) Signup, and (3) Exit App. The source code below contains the `main()` application program which interacts with a user of the application. Don't be frightened by the source code, we'll revist it later. For now, please proceed to the second section.

In [0]:

```
# Import python libraries
import sqlite3
import sys

import warnings   # This suppress warning messages
warnings.filterwarnings('ignore')

# Import AddressBook main classes, i.e., Object Oriented Programming (OOP) concept.
from manage_users import ManageUsers
from manage_contacts import ManageContacts


# Define main application function
def main():
    """
    This is the main application function.
    :return:
    """

    # Initialize menu input variable
    choice_menu = 1

    while choice_menu != 0:
        # Main menu - login / signup page
        print("""
        --- Welcome to AddressBook Application platform ---

        1. Login
        2. Signup
        0. Exit App
        """)

        # Get user's menu input key value.
        input_menu = int(input("Please enter your choice from 0 to 2: "))

        # Process user's request.
        if input_menu == 1:
            # Process login request.
            response_login = login_menu()

            # Allow user to use AddressBook App if login is successful.
            if response_login:
                # Set the username as part of the current session.
                # It will be inserted in contacts table directly.
                username_session = response_login['username']

                # Greet user nicely.
                print("\nWelcome, ", response_login['first_name'], '!')

                # Submenu - create, read, update and delete contacts
                choice_submenu = 1

                while choice_submenu != 0:
                    print("""
                    1. Add new contact
                    2. Show my contacts
                    3. Search contact
                    4. Update contact
                    5. Delete contact
```

```
                    0. Logout
                    """)

                    # Get user's submenu input key value.
                    input_submenu = int(input("Please enter your choice from 0 to 5:

                    # Process submenu request
                    if input_submenu in [1, 2, 3, 4, 5]:
                        contact_submenu(username_session=username_session,
                                        input_submenu=input_submenu)
                    elif input_submenu == 0:
                        # Logout
                        print("Logging out...")
                        sys.exit()
#                         exit()
                    else:
                        pr
                        int("Invalid submenu input value. Please reenter values from
                else:
                    print("Login Failed. Please try again!")
        elif input_menu == 2:
            # Signup new user menu
            response_signup = signup_menu()

            if response_signup:
                print("""
                Yay, signup is successful!
                You're now ready to use AddressBook App. Please login.
                """)
            else:
                print("Signup Failed. Please try again!")
        elif input_menu == 0:
            # Close Address Book App menu
            print("Exiting...")
            sys.exit()
#             exit()
        else:
            print("Invalid menu input value. Please reenter values from 0 to 2!")


def login_menu():
    """
    This function allows a user to login.
    """

    print("""
    --- Welcome to login page ---
    """)

    username = input("Please enter your username: ")
    password = input("Please enter your password: ")

    user = ManageUsers()
    response_login = user.login(username, password)

    if response_login:
        return response_login
    else:
        return False
```

```
def signup_menu():
    """
    This function facilitates user registration.
    """

    print("""
    --- Welcome to signup page ---
    """)

    # Create ManageUsers object.
    new_user = ManageUsers()

    # Get registration info from the user.
    username = input("Please enter your username: ")
    first_name = input("Please enter your first name: ")
    last_name = input("Please enter your last name: ")
    password = input("Please enter your password: ")
    password_again = input("Please reenter your password again: ")

    # Process user's input values.
    response_signup = new_user.create_user(username, first_name, last_name,
                                           password, password_again)

    # Return result
    if response_signup:
        return True
    else:
        return False


def contact_submenu(username_session, input_submenu):

    if input_submenu == 1:
        """
        Add a new contact
        """

        print("""
        --- Adding new contact... ---
        """)

        # Create ManageContacts object.
        new_contact = ManageContacts()

        # Get contact's details (e.g., first name, last name, address, etc.
        first_name = input("Please enter contact's first name (required): ")
        last_name = input("Please enter contact's last name (required): ")
        address = input("Please enter contact's address (required): ")
        phone = input("Please enter contact's phone number (required): ")
        email = input("Please enter contact's email address (optional): ")
        notes = input("Any notes? (optional): ")

        # Add contact to database.
        response_create_contact = new_contact.create_contact(first_name, last_name,
                                                             phone, email, notes, us

        # Retun result.
        if response_create_contact:
            new_contact.read_contacts(username=username_session)

            return True
```

```
        else:
            return False
    elif input_submenu == 2:
        """
        Show all contacts
        """

        contact = ManageContacts()
        contact.read_contacts(username=username_session)
    elif input_submenu == 3:
        """
        Search contact
        """

        # Get what to search from user
        first_name_search = input("Please enter first letter of your contact first r

        contact = ManageContacts()
        contact.search_contact(username=username_session,
                               first_name_start=first_name_search)
    elif input_submenu == 4:
        # TODO complete update functionality
        pass
    elif input_submenu == 5:
        # TODO complete delete functionality
        pass
```

How to signup for AddressBook App?

Suppose there's a new user of the AddressBook application whose name is `Steve Jobs`. A registration is first required to use the application. How can he do that? By running the `main()` program as shown below, the app displays three options.

Steve was directed to the signup page when he entered a value `2` in the terminal. The signup page asks Steve to enter the following data about himself, e.g., username, first name, second name, and password. For a validation reason, the app requests users to re-enter their password again.

Once the password is validated, the app returns registration confirmation. Steve was registred successfully. The app will then suggest the user to proceed to login with the following message: `You're now ready to use AddressBook App. Please login.`

By typing `0`, Steve decided to exit the application for now. Because of that the app displays this message on screen: `Exiting...`

In [0]:

```
# Run main application program for a signup
main()
```

```
        --- Welcome to AddressBook Application platform ---

        1. Login
        2. Signup
        0. Exit App

Please enter your choice from 0 to 2: 2

    --- Welcome to signup page ---

Please enter your username: steve
Please enter your first name: steve
Please enter your last name: jobs
Please enter your password: steve123
Please reenter your password again: steve123

                Yay, signup is successful!
                You're now ready to use AddressBook App. Please login.


        --- Welcome to AddressBook Application platform ---

        1. Login
        2. Signup
        0. Exit App

Please enter your choice from 0 to 2: 0
Exiting...


An exception has occurred, use %tb to see the full traceback.

SystemExit
```

How to login to AddressBook App?

Steve was a best friend of Linus Torvalds (https://en.wikipedia.org/wiki/Linus_Torvalds), and decided to use the AddressBook app to keep Linus in his contacts. How could he do that? He could do this by first logging into the app.

As shown in the source code below, he typed `1` in the terminal which prompted him to enter his username and password. After a successful login, the AddressBook app would greet him with a big smile. Moreover, the app displays him with a contact menu. In the menu, five operations related to contacts can be performed, i.e., (1) add a new contact, (2) show contacts, (3) search for a contact, (4) update a contact, and (5) delete a contact.

Steve then selected the first option by typing `1` in the terminal. This operation prompted him to enter the following data about Linus: first name, last name, address, phone number, email, and any notes. It seems Linus was successfully added to the contact database. The app then displays all contacts of Steve, in this case only Linus.

Steve logged out from the app by typing `0` in the terminal. Don't worry, he will come back sometime later to add his good-old-friends into his contact list :-)

In [0]:

```
# Run the main application program for a login
main()
```

```
        --- Welcome to AddressBook Application platform ---

        1. Login
        2. Signup
        0. Exit App

Please enter your choice from 0 to 2: 1

    --- Welcome to login page ---

Please enter your username: steve
Please enter your password: steve123

Welcome,  steve !

                    1. Add new contact
                    2. Show my contacts
                    3. Search contact
                    4. Update contact
                    5. Delete contact
                    0. Logout

Please enter your choice from 0 to 5: 1

        --- Adding new contact... ---

Please enter contact's first name (required): linus
Please enter contact's last name (required): torvaldus
Please enter contact's address (required): linux street
Please enter contact's phone number (required): +1 415 723 9709
Please enter contact's email address (optional): info@linuxfoundation.
jp
Any notes? (optional): linux founder

            --- List of your contacts stored in AddressBook App ---

(5, 'linus', 'torvaldus', 'linux street', '+1 415 723 9709', 'info@lin
uxfoundation.jp', 'linux founder', 'steve')

                    1. Add new contact
                    2. Show my contacts
                    3. Search contact
                    4. Update contact
                    5. Delete contact
                    0. Logout

Please enter your choice from 0 to 5: 0
Logging out...

An exception has occurred, use %tb to see the full traceback.

SystemExit
```

The Magic behind AddressBook App: The ManageUsers and ManageContacts objects

The two main object-oriented (https://en.wikipedia.org/wiki/Object-oriented_programming) classes that are used by the AddressBook app are `ManageUsers` and `ManageContacts`. I implemented CRUD (https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) operations in both classes of the application for processing the database and for displaying data to users.

The ManageUsers object

The `ManageUsers` class is designed to create (i.e., register or signup) new users, display all users, update and delete user's records from database. In addition to the four CRUD operations, I added `login()` functionality into the `ManageUsers` class to process user's login into AddressBook app platform.

The `create_user()` function calls two sub-functions, i.e., `validate_username()` and `validate_password()` functions, for validating username (in case a username is alrady taken by old users) and for validating a password (in case there's a missmatch between the first password and the reentered password).

In [0]:

```
import sqlite3
from database_connection import DatabaseConnection


class ManageUsers:
    """
    This class manages users to add user, show users, update and delete user.
    In the class constructor, i.e., __init__, a database connection is established.
    """

    def __init__(self):

        # connect to database
        self.db = DatabaseConnection()
        self.db_connect = self.db.connect_database()

        # create a Cursor() method from established database connection
        self.cur = self.db_connect.cursor()

    # CRUD operations, i.e., 'C' in CRUD stands for Create, 'R' for Read,
    # 'U' for Update, and 'D' for Delete.
    def create_user(self, username, first_name, last_name,
                    password, password_again):

        # Validation step: check if username is already taken
        check_username = self.validate_username(username)

        if check_username:
            print("""
            WARNING MESSAGE - The username exists. Please try again.
            """)

            return False
        else:
            # Validation step: password validation
            check_password = self.validate_password(password, password_again)

            if check_password:

                # Register the user
                try:
                    query_insert_user = """
                    INSERT INTO users(username, first_name, last_name, password)
                    VALUES(?, ?, ?, ?)
                    """

                    params = (username, first_name, last_name, password)

                    self.cur.execute(query_insert_user, params)
                    self.db_connect.commit()

                    return True
                except sqlite3.Error as err:
                    print('Err: ', err.message)

                    return False
            else:
                print("""
                Password mismatch. Please try again!
```

```
            """)

            return False

    def read_users(self):

        try:
            query_show_users = "SELECT * FROM users"

            self.cur.execute(query_show_users)
            data = self.cur.fetchall()

            print("""
            --- List of users for AddressBook App ---
            """)

            for row in list(data):  # NOTE The list() function forces print()
                                    # to print all records returned by fetchall().
                print(row)

            return True
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False

    def update_user(self, username):
        pass  # TODO complete update_user() function

    def delete_user(self, username):
        pass  # TODO complete delete_user() function

    def login(self, username, password):

        try:
            query_login_credentials = """
            SELECT * FROM users
            WHERE username = ? AND password = ?
            """

            params = (username, password)
            self.cur.execute(query_login_credentials, params)
            data = self.cur.fetchall()  # NOTE sqlite3 returns a tuple, e.g., (1, 't

            if data:
                # store user credentials as a dictionary
                result = {'id': data[0][0],
                          'username': data[0][1],
                          'first_name': data[0][2],
                          'last_name': data[0][3]}

                return result
            else:
                print("""
                WARNING MESSAGE - Login failed. Please try again!
                """)

                return False
        except sqlite3.Error as err:
            print('Err: ', err.message)
```

```
            return False

    def validate_username(self, username):

        try:
            query_find_user = "SELECT * FROM users WHERE username = ?"

            params = (username,)
            self.cur.execute(query_find_user, params)
            data = self.cur.fetchone()

            if data:
                return True
            else:
                return False
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False

    def validate_password(self, password, password_again):
        # TODO Method validate_password might be static - read and understand
        if password != password_again:
            return False
        else:
            return True
```

The ManageContacts object

Like the `ManageUsers` class, the `ManageContacts` class is designed based on CRUD (https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) operations for processing user's contacts. The `search_contact()` function is designed to process user's search for a contact.

In [0]:

```
import sqlite3
from database_connection import DatabaseConnection


class ManageContacts:
    """
    This class manages user's contacts to add contact, show contacts, search contact
    In the class constructor, i.e., __init__, a database connection is established.
    """

    def __init__(self):

        # connect to database
        self.db = DatabaseConnection()
        self.db_connect = self.db.connect_database()

        # create a Cursor() method from established database connection
        self.cur = self.db_connect.cursor()

    # CRUD operations, i.e., 'C' in CRUD stands for Create, 'R' for Read,
    # 'U' for Update, and 'D' for Delete.
    def create_contact(self, first_name, last_name, address,
                       phone, email, notes, username):

        try:
            query_insert_contact = """
            INSERT INTO contacts(first_name, last_name, address,
                                 phone, email, notes, username)
            VALUES (?, ?, ?, ?, ?, ?, ?)
            """

            params = (first_name, last_name, address,
                      phone, email, notes, username)

            self.cur.execute(query_insert_contact, params)
            self.db_connect.commit()

            return True
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False

    def read_contacts(self, username):

        try:
            query_show_contacts = """
            SELECT * FROM contacts
            WHERE username = ?
            """

            params = (username,)

            self.cur.execute(query_show_contacts, params)
            data = self.cur.fetchall()

            print("""
            --- List of your contacts stored in AddressBook App ---
            """)
```

```
            for row in list(data):  # NOTE The list() function forces print()
                                    # to print all records returned by fetchall().
                print(row)

            return True
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False

    def update_contact(self, username, first_name, phone):
        # NOTE first_name and phone makes unique operation
        # TODO complete update_contact() function
        pass

    def delete_contact(self, username, first_name, last_name):
        # TODO In create_contacts() function implement defensive code to not allow u
        # TODO a new contact with existing first_name and last_name
        pass

    def search_contact(self, username, first_name_start):
        try:
            query_search = """
            SELECT * FROM contacts
            WHERE username = ? AND first_name LIKE ?
            """

            first_name_pattern = first_name_start + '%'
            params = (username, first_name_pattern)

            self.cur.execute(query_search, params)
            data = self.cur.fetchall()

            if data:
                print("""
                --- Your search result: ---
                """)

                for row in list(data):
                    print(row)

                return True
            else:
                print("No contact found!")

                return False
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False
```

Database powered by Sqlite3 in Python

I used SQLite (https://en.wikipedia.org/wiki/SQLite) database to store data about AddressBook application users and their contacts. The SQLite is a standard library and comes with Python 3.6 (https://en.wikiversity.org/wiki/Python_Programming/Databases); thus, you simply import it (like `import sqlite3`), when you use it for the first time.

Creating Database for AddressBook App using Sqlite3 in Python

As shown in the script below, I created a database called `address_book.db` . In this database, I created two tables, namely `users` and `contacts` , for storing users profile and their contacts, respectively. The primary keys of these tables are `user_id` and `contact_id` , respectively.

Note that the `username` field is defined in both tables. The reason this field is defined in the `contacts` table is for creating a relationship with `users` table.

In [0]:

```
import sqlite3

# Create database
db = sqlite3.connect('address_book.db')
cur = db.cursor()

# Create users table (i.e., used for login)
# - user_id is a primary key here.
cur.executescript("""
CREATE TABLE IF NOT EXISTS users(
user_id INTEGER PRIMARY KEY,
username VARCHAR(20) NOT NULL,
first_name VARCHAR(20) NOT NULL,
last_name VARCHAR(20) NOT NULL,
password VARCHAR(20) NOT NULL
)
""")

# Create contacts table
# - contact_id is a primary key here.
# - username is a foreign key here.
cur.executescript("""
CREATE TABLE IF NOT EXISTS contacts(
contact_id INTEGER PRIMARY KEY,
first_name VARCHAR(20) NOT NULL,
last_name VARCHAR(20) NOT NULL,
address VARCHAR(50) NOT NULL,
phone VARCHAR(20) NOT NULL,
email VARCHAR(20),
notes VARCHAR(50),
username VARCHAR(20) NOT NULL
)
""")
```

The DatabaseConnection object

The `DatabaseConnection` class handles database connection for AddressBook app. I passed database name of the application via the constructor function, i.e., `database_name='address_book.db` . In case the database name of the application changes for some reason, we make a single change in the constructor function.

Both `ManageUsers` and `ManageContacts` classes calls the `DatabaseConnection` class in their constructor functions to setup database connection of our AddressBook app (see above). The `close_database()` function is not completed; that's why I used `PASS` method which is simply a place-

holder which does nothing.

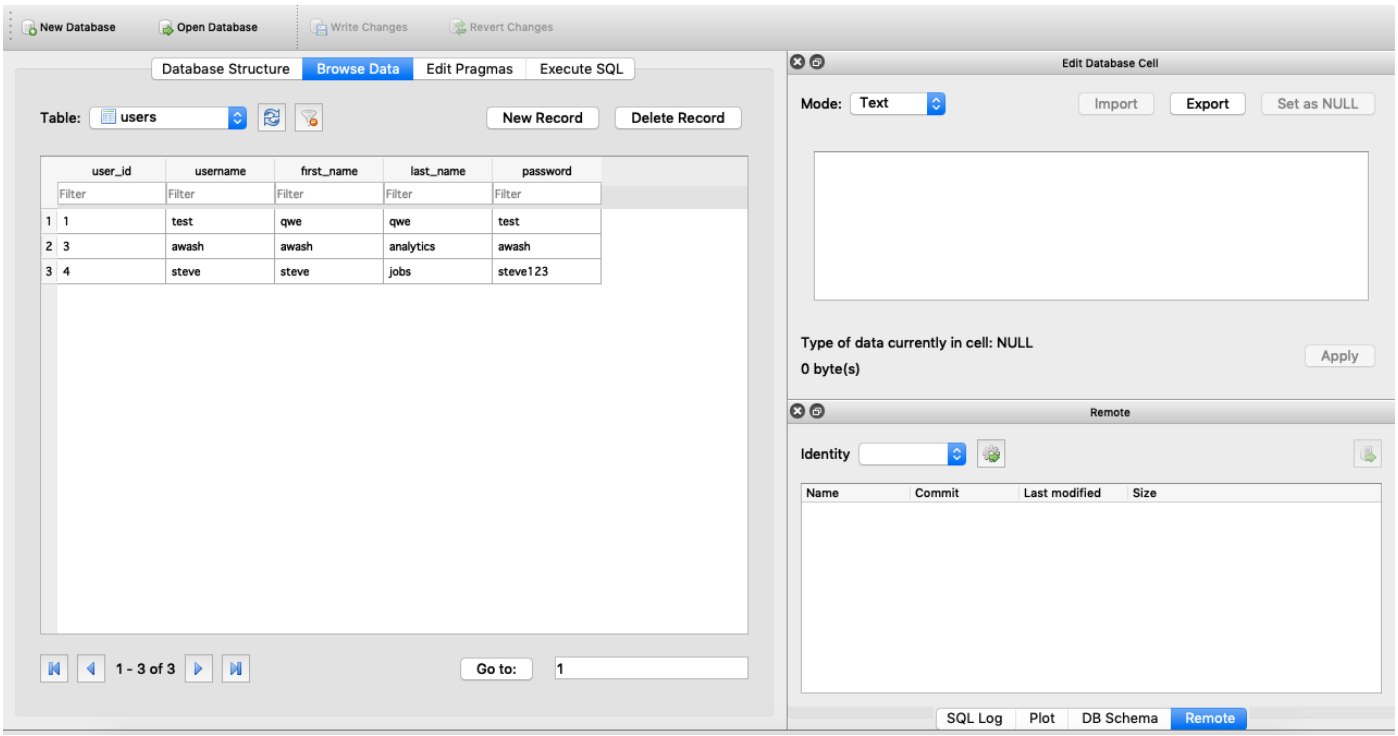
In [0]:

```
import sqlite3


class DatabaseConnection:

    def __init__(self, database_name='address_book.db'):
        self.database_name = database_name
        self.db = None

    def connect_database(self):
        try:
            self.db = sqlite3.connect(database=self.database_name)

            return self.db
        except sqlite3.Error as err:
            print('Err: ', err.message)

            return False

    def close_database(self):
        pass  # TODO complete close_database() function
```

How to View Sqlite3 Database?

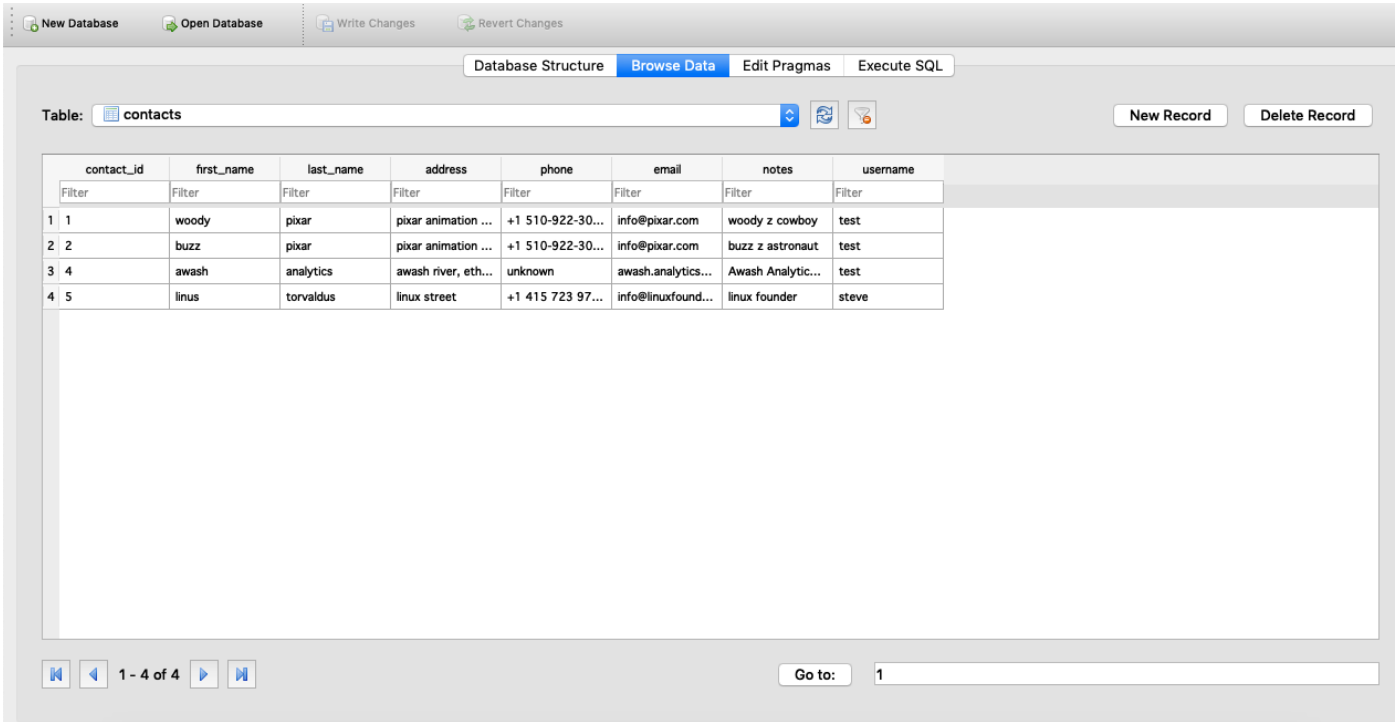
You can view and process your application's database using **DB Browser for SQLite** tool. You can download this tool using this **link** (https://sqlitebrowser.org/).

The screenshot below shows you the `users` table of the AddressBook app. There're three users of the AddressBook app, namely `test`, `awash` and `steve`.



In the second screenshot shown below, all records stored in the `contacts` table is displayed. There're are

four contacts, among which the first three belong to a user called `test` and the last contact belongs to `steve`.



Conclusion

I hope the AddressBook application could give you some idea how to build your own application using Python. For building such application, knowledge of **Object-oriented programming** is a key to conceptualize the physical world and to simplify a complex problem into smaller pieces (e.g., users and contacts).

Let me know if you would like to make your hand dirty using the AddressBook applcation. I can help you setting up the environment for you on your machine. Good luck!

Ps. In case you would like to build your own application following the same logic as used in the AddressBook app, check **Awash Analytics** (http://awash-analytics.info/) website for similar projects.

In [0]: