Report: Address Book CLI

Author: Robert Rash

This is a semester project for Software Security (CSE 5359), wherein I've implemented a command line address book program. The overarching goal of this assignment is input validation: ensure that only those names and phone numbers that meet the desired specifications are entered into the database. This document outlines the program's architecture, the "desired specifications", and discusses some pros and cons of these decisions.

Program Architecture

`src/address_book.py`

Performs all of the Create Read Update Destroy (CRUD) operations related to the address book, though there's no update feature. Acts as a bridge between user input received from `main.py`, validation operations defined in `validator.py`, and the database.

`src/db_utils.py`

Establishes a singleton connection with the database, creating a SQLite database file if necessary. Also defines a context manager for atomic database transactions.

`main.py`

Contains the top-level command line argument parsing system. Parses command line arguments and passes those values to the underlying address book system.

`src/models.py`

Defines a SQLAlchemy ORM model for interacting with the database. Contains fields for a name and a phone number, each of which map to database columns with a `UNIQUE` constraint.

`src/regular_expressions.py`

Defines compiled, reusable regular expression objects intended for use anywhere. In this program, these

objects are used solely in `src/validator.py`.

`src/validator.py`

Defines a series of functions for normalizing and validating user-provided names and phone numbers.

Assumptions Regarding Acceptable Name and Phone Number Format Standards

When it comes to names and telephone numbers, there are myriad forms that each can take. This fact has been a constant struggle for programmers for ages, both from a functional perspective and a security perspective. It is perhaps impractical to attempt to come up with a "silver bullet" solution to this problem. A better approach may be to devise a range of acceptable forms for each of these categories. For the sake of simplicity, this is what has been done here.

Names

Names vary significantly the world over. They can take all kinds of shapes and forms depending on the origin. It would be a near insurmountable task to truly "validate" a name. A more considerate approach may be to simply blacklist certain characters that are likely to not appear in a real name and that in some cases may appear in malicious input. There are, indeed, exceptions to this, but those are considered exceedingly rare edge cases.

This blacklist takes the form of excluding a character class containing common punctuation. There is some punctuation, however, that is not included in the blacklist, such as characters in the range `[.,"-']`, which are all used commonly in names or in formatting names.

This is an incredibly permissive policy; it allows almost any kind of data in. However, in addition to filtering out symbols that are commonly used in injection attacks, input is also escaped before being entered into the database. This provides a very robust system for filtering out attempts at malicious injection through input fields.

The regular expression used for names is as follows:

```
'''
^
    [^\0\r\n\t!@#$%^&*_+=(){}\[\]<>\\|;:\/?]+
$
'''
```

This regular expression matches any line that does **NOT** contain any of the characters `\0\r\n\t!@#$%^&*_+=(){}[]<>\|;:/?`.

Phone numbers

For this project, acceptable phone number formats will follow

- (1) the standards set forth by the North American Numbering Plan (NANP) for numbers inside this jurisdiction and
- (2) the standards set forth in ITU E.164 for numbers outside of the NANP jurisdiction.

Normalization

Phone number inputs pass through an initial normalization phase wherein characters that are not numbers or the `+` character are removed. This provides a (somewhat) more standardized input for the following steps, meaning that the later regular expressions are greatly simplified.

North American Numbering Plan (NANP)

The North American Numbering Plan includes 25 regions in twenty countries in North America, including the United States and its territories, Canada, and a significant portion of the Caribbean.

A proper NANP phone number consists of these sections:

```
+1          (optional) NANP international calling code

1           (optional) domestic use only, and never in
            combination with the previous section;
            part of a change made to increase the pool of
            numbers in a given area code

XXX         area code -- ranges: [2-9] for first digit,
            [0-8] for the middle digit, and [0-9] for
            the final digit *

XXX         central office code -- ranges: [2-9] for
            first digit, [0-9] for next two digits **

XXXX        subscriber code -- ranges: [0-9] for all
            four digits
```

* when the last two digits are the same (i.e. NXX where X is in [0-8]), this code is known as an easily

recognizable code (ERC), which are used to designate special services (e.g. 888 for toll-free service)

** the last two digits may not be the same if they are both '1'

Acceptable Examples:

```
# valid 10-digit
2342355678
234-235-5678
234 235-5678
(234) 235-5678
234 235 5678
234.235.5678

# valid 10-digit w/ international code, w/o '+'
12342355678
1 234-235-5678
1 234 235-5678
1 (234) 235-5678
1 234 235 5678
1 234.235.5678

# valid 10-digit w/ international code, w/ '+'
+12342355678
+1 234-235-5678
+1 234 235-5678
+1 (234) 235-5678
+1 234 235 5678
+1 234.235.5678
```

Unacceptable Examples:

```
# invalid 7-digit *
2355678
235-5678
235 5678
235.5678

# invalid 7-digit w/ extension * **
2355678x4321
235-5678 extension 4321
235 5678 ext4321
235.5678#   4321

# invalid 10-digit w/ extension **
2342355678x4321
234-235-5678 extension 4321
234 235-5678 ext4321
(234) 235-5678 # 4321
234 235 5678 x 9
234.235.5678 extension 5900000

# invalid -- area code must start with [2-9]
123-234-5678

# invalid -- office code (two trailing numbers == '1')
234-911-5678

# invalid -- office code (starts with '1')
314-159-2653
```

* As a matter of policy, seven-digit NANP phone numbers are considered unacceptable. Outside of a given area code, seven-digit NANP phone numbers are essentially useless without the context provided by an area code, since those seven-digit numbers would only connect locally.

** Extensions are not covered in this program. Technically speaking, extensions are an extra instruction on top of the resource that a phone number represents, so arguably they are not part of the phone number.

It should be noted that by no means are the above example lists of exhaustive. There are so many possible test cases that it would be difficult to create an exhaustive list.

The regular expression used for NANP numbers is as follows:

```
'''
^
    (?:\+?1)?
    ([2-9]1[02-9]|[2-9][02-8]1|[2-9][02-8][02-9])
    ([2-9]1[02-9]|[2-9][02-9]1|[2-9][02-9]{2})
    ([0-9]{4})
$
'''
```

This regular expression allows an optional `+1` or `1`, with the rest consisting of capture groups conforming to each of the NANP guidelines described above.

ITU E.164

Numbers conforming to this standard are required to input the number with a `+` followed by the country calling code (a list of of these calling codes can be found at the following link).

E.164 phone numbers are a maximum of 15 digits, where the first 1-3 digits are the country calling code and the following digits are the national number, ranging from 4-14 digits based on the location and the length of the country calling code.

The ITU E.164 standard is described here (PDF download).

Acceptable Examples:

```
# valid -- '+' included, proper number length
+44 300 222 0000
+49 40 338036
```

Unacceptable Examples:

```
# invalid -- no '+'
44 300 222 0000
49 40 338036

# invalid -- proper '+' and country code, but too short number
+210 031
```

The regular expression used for ITU E.164 numbers is as follows:

```
'''
^
    (?:\+{1})
    (20|210|211|212|213|214|215|216|217|218|219|220|221|222|223|224|225|226|227|22
    (\d{4,14})
$
'''
```

This regular expression has a non-capturing group to catch the requisite `+`, a capturing group containing all known country calling codes excluding `+1` (scraped from the Wikipedia page for country calling codes), and a final capturing group containing the remaining numeric characters.

Pros and Cons

Names

Pros:

- allows for great flexibility for a type of data that has next to no standardization
- manages to filter out a significant portion of malicious input
- relatively simple to implement
- simple implementation results in better regex performance

Cons:

- (potentially) cascading effects
    - if a user inputs non-standard data that manages to pass the regex, potential functionality based on the name may be negatively impacted (e.g. if this program were to be implemented into a webpage, non-standard input may impact that webpage in unknown ways)

Phone numbers

Pros:

- accepting NANP phone numbers in addition to E.164 (a widely-applicable phone number format) covers the vast majority of worldwide phone numbers
- normalization as a first step greatly increases flexibility of user input
    - allows for almost any potential phone number format to be entered as long as the basic qualifiers are included (e.g. '+', country code, and number for E.164)

- normalization as a first step ***greatly*** simplifies regex implementation
- simpler regex implementation results in better regex performance

Cons:

- method of normalization may allow for unintended / fake phone numbers to be extracted from garbage data
    - normalization strips all characters that aren't numbers or the '+' sign -- this could result in extracting a series of numbers embedded in a string of garbage data that were not intended to be a phone number but can still pass validation

- regex validation may validate fake numbers
    - there are a great many possible numbers that may pass the validation but are in fact not real phone numbers. To check for a valid phone number in this way would be entirely outside of the scope of this project, though