P05 – Dictionaries and JSON

1. Vocabulary Trainer

Write a program that helps you to memorize English words (e.g., technical terms).

The program has two tasks:

- Load the translations from a file into a Python dictionary. To start you can use the provided file **`dictionary.txt`**, which you can extend by any words you want to memorize.

```
# Python dictionary that contains the translations loaded from the file
{'key': ['Schluessel', 'Taste'], 'expression': ['Ausdruck']}
```

- While the user did not know all translations correctly, randomly choose an English word in the dictionary and ask the user for its translation. If the translation is correct you can remove the entry from the dictionary.

Hint: For one English word you can have multiple German translations. Note how this is reflected in the data structure above.

2. Phone Book

Write a program that manages telephone numbers. It should be possible to

- add a new contact (name and number)
- remove a contact
- lookup a phone number for a given name
- save the contacts into a file
- load the contacts from a file

2.1 Basic Phone Book

You can use the template provided below (or the file **`p05_phone_book_template.py`**). Implement the missing function bodies without altering the interface they comprise in conjunction with the given functions calls.

```
def print_numbers(numbers):
    #Todo
def add_number(numbers, name, number):
    #Todo
def lookup_number(numbers, name):
    #Todo
def remove_number(numbers, name):
    #Todo
```

```
def load_numbers(numbers, filename):
    #Todo
def save_numbers(numbers, filename):
    #Todo

def print_menu():
    print '1. Print Phone Numbers'
    print '2. Add a Phone Number'
    print '3. Remove a Phone Number'
    print '4. Lookup a Phone Number'
    print '5. Load numbers'
    print '6. Save numbers'
    print '7. Quit'
    print

phone_book = {}
menu_choice = 0
print_menu()
while True:
    menu_choice = int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(phone_book)
    elif menu_choice == 2:
        print "Add Name and Number"
        name = raw_input("Name: ")
        phone = raw_input("Number: ")
        add_number(phone_book, name, phone)
    elif menu_choice == 3:
        print "Remove Name and Number"
        name = raw_input("Name: ")
        remove_number(phone_book, name)
    elif menu_choice == 4:
        print "Lookup Number"
        name = raw_input("Name: ")
        print lookup_number(phone_book, name)
    elif menu_choice == 5:
        filename = raw_input("Filename to load: ")
        load_numbers(phone_book, filename)
    elif menu_choice == 6:
        filename = raw_input("Filename to save: ")
        save_numbers(phone_book, filename)
    elif menu_choice == 7:
        break
    else:
        print_menu()
```

2.2 Phone Book Pro

Implement at least two of the following extensions:

- Make it possible to save multiple numbers for the same person
- Make it possible to find people with typos
- Make it possible to find all that start with a letter

- Make it possible to save more data that belongs to that person, birthday, address, …
- Make it possible to save the data into an excel sheet

3. Transport at opendata.ch

Suppose the marketing department of your company would like to launch a campaign for a new product series. It would like to set up booths at train stations to reach as many people as possible. In order to prioritize where to set up booths, they would like to know…

- how many people can be reached
- between 4pm and 7pm on a weekday
- in six of the largest cities in Switzerland: Zurich, Geneva, Basel, Lausanne, Bern, and Winterthur.

The assumption is that the more trains depart, the more people will be there.

A web service at http://transport.opendata.ch provides information on the Swiss public transit system through a **`REST`** interface, e.g. for all timetable information of public transit in Switzerland. With a **`HTTP GET`** request to one of the three resources **`/locations`**, **`/connections`** or **`/stationboard`**, a **`JSON`** response is returned containing objects describing locations, stops, journeys and others. Detailed information is provided on the website.

Using this web service you should write a script to answer the following problem:

- Use http://transport.opendata.ch/stationboard to retrieve the number of connections in each of the six cities between 4pm - 7pm on a weekday (e.g. Wednesday)
- Come up with a *simple* heuristic of how to assign to each departing train a certain number of people at the station at this time
- Aggregate the numbers to answer the question of the marketing department by presenting the quantities for all six cities, and stating the winner

To get started, you have the following script (also in **`p05_transport_opendata_ch_template.py`**) available, which does the job of telling an end user which trains depart in a given city and a given hour.

- Refactor this script in order to serve your needs in this task
- You can assume that the choice of trains in the script (only S-Bahn) is also well-suited for your problem

```
import urllib
import json
from datetime import datetime, timedelta, date
```

```
api_url = "http://transport.opendata.ch/v1/stationboard"

# User input
city_name = raw_input("Looking up trains. Which city? ")
start_time = raw_input("What time? Type like 22:30 ")

# create datetime objects for start and stop (one hour later)
start = datetime.combine(date.today(),
                         datetime.strptime(start_time,'%H:%M').time())
stop = start + timedelta(hours=1,minutes=1)
# Prepare http get variables as dictionary
get_values = {}
get_values["station"] = city_name
get_values["datetime"] = start.strftime('%Y-%m-%d %H:%M')#start time
get_values["transportations[]"] = 's_sn_r'# just get the s-bahn



try:
    got_one_hour = False
    num_requests = 0
    count = 0

    print 'Connections leaving from',city_name
    print 'between',start.strftime('%H:%M'),"and",stop.strftime('%H:%M')

    while not got_one_hour:
        url_values = urllib.urlencode(get_values)
        full_url = api_url + "?" + url_values
        response = urllib.urlopen(full_url)
        json_tree = json.loads(response.read())

        if 'stationboard' in json_tree:
            num_requests +=1
            all_journeys = json_tree['stationboard']
            for journey in all_journeys:
                dep_time = datetime.strptime(journey['stop']['departure'][:16],'%Y-%m-
%dT%H:%M')
                if dep_time < stop:
                    count +=1
                    print '  ',journey['name'],'-to-', journey['to'], ' at',
dep_time.strftime('%H:%M')
                else:
                    got_one_hour = True
                    break
            get_values["datetime"] = (dep_time + timedelta(minutes=1)).strftime('%Y-%m-%d
%H:%M')
        else:
            print 'HTTP response seems to be empty'
            break
    print 'found',count,'connections using',num_requests,'requests'

except Exception, detail:
    print "Couldn't connect to the transport api.", detail
```

Note that the **`/stationboard`** resource returns about 40 journeys for a given start time. Therefore, the script checks the departure time of the journeys to determine if additional requests are needed to obtain information on the full hour.

Tipp: The http://transport.opendata.ch web service is an unofficial interface to the SBB data. For commercial projects, the downloadable flat files at http://www.fahrplanfelder.ch/fahrplandaten/ might be very interesting. Additionally, SBB is open to all kinds of questions and requests related to data via data@sbb.ch.