INF1340
Introduction to Information Systems
Instructor: Susan Sim
Fall 2015

Assignment #3
Deadline: Wednesday, 16 December 2015, 11:59pm

This assignment involves the creation of programs that involve working with functions, strings, and lists. Please make an effort to do the assignment without copying solutions from the Internet.

This assignment is worth 30% of the final mark. The criteria for evaluation are found at the end of the handout. It contains **2 separate exercises**. It is scored out of 100.

Starter Files

For each exercise, two files have been provided. One contains the program code and can be run using the Python interpreter in PyCharm. The other contains test code and can be run using py.test.

The files can be accessed on github at: `https://github.com/benevolentprof/inf1340_2015_asst3.git`

Submitting Your Assignment

Each team should work on a single fork of the repo.

On Blackboard, a "test" has been created where you can submit your github URL and names of team members. Only one member of each team needs to complete the test. The last commit to the github repository before the due date will be graded.

Exercise 1: Database Management System Redux

This exercise continues the DBMS example from Assignment 2. The three table operations to be implemented are selection, projection, and cross product.

Functions to Add

`selection(table1, function)`
Returns the table that is the result of applying `function` to the `table1`. function is expected to operate on the data rows of table1. If the result of a operation is an empty table, the function should return `None`.

Here's an example table.

```
EMPLOYEES = [["Surname", "FirstName", "Age", "Salary"],
             ["Smith", "Mary", 25, 2000],
             ["Black", "Lucy", 40, 3000],
             ["Verdi", "Nico", 36, 4500],
             ["Smith", "Mark", 40, 3900]]
```

Here's an example function.

```
def filter_employees(row):
    """
    Check if employee represented by row
    is AT LEAST 30 years old and makes
    MORE THAN 3500.
    :param row: A List in the format:
        [{Surname}, {FirstName}, {Age}, {Salary}]
    :return: True if the row satisfies the condition.
    """
    return row[-2] >= 30 and row[-1] > 3500
```

The result of `select(EMPLOYEES, filter_employees)` would be

```
[["Surname", "FirstName", "Age", "Salary"],
["Verdi", "Nico", 36, 4500],
["Smith", "Mark", 40, 3900]]
```

`projection(table1, attributes)`

Reduces a table to the columns named in attributes. `attributes` is a list, so it can contain any number of column names. If a column is in the `attributes` list, but not `table1`, raise an error. For example, `project(GRADUATES, [“Surname”])` would return:

```
       [[“Surname”],
       [“Robinson”],
       [“O’Malley”],
       [“Darkes”]]
```

`cross_product(table1, table2)`

The cross product function combines two tables by matching every row from `table1` with every row in `table2`. If the result of a operation is an empty table, the function should return `None`.

```
R1 = [["Employee", "Department"],
      ["Smith", "sales"],
      ["Black", "production"],
      ["White", "production"]]

R2 = [["Department", "Head"],
      ["production", "Mori"],
      ["sales", "Brown"]]
```

```
result = [[“Employee”, “Department”, “Department”, “Head”],
          [“Smith”, “sales”, “production”, “Mori”],
          [“Smith”, “sales”, “sales”, “Brown”],
          ["Black", “production”, “production", "Mori"],
          [“Black", “production”, "sales", "Brown"],
          ["White", “production”, "production", "Mori"],
          ["White", “production”, "sales", "Brown"]]
```

Testing

Add your own test cases to test_exercise3.py. Include tables with various schema and produce a variety of results with the table operations.

Exercise 2: Papers, Please!

Papers, Please is a hit indie game from 2013 (http://papersplea.se/). In the game, you play an immigration inspector for the communist state of Arstotzka. Your job is to control the flow of people entering the country, using only documents provided by the travellers and directives from the ministry. You need to let in legitimate immigrants and visitors, and keep out the criminals and ne’er-do-wells.

The game is surprisingly fun– who knew a dystopian document thriller was a genre? It is challenging for human beings, because the player needs to remember a lot of details and make decisions quickly. Fortunately, this is the kind of thing that computers do well.

This assignment involves the creation of a program that will act as an immigration inspector for the country of Kanadia. It will give you some experience with business rules and partitioning code into functions.

Included in the initial code are a number of JSON files containing information from travellers seeking entry and the latest directives from the Ministry.

Specification

Your program will receive an entry record and must output one of the three following choices.

- Accept
- Reject
- Quarantine

Every entry record must contain information.

- First Name
- Last Name
- Birth Date (YYYY-MM-DD)
- Passport Number (five groups of five alphanumeric characters separated by dashes)
- Home (Location)
- From (Location)
- Reason for Entry (returning or visit)

Some entries will have additional information.

- Via (Location)
- Visa

Each location consists of three fields.

- Country Code
- Country Name

Each visa will have two fields.

- Date (YYYY-MM-DD)
- Code (five groups of five alphanumeric characters separated by dash)

The Ministry provides lists that have the following format.

- City
- Region
- Country Code

Each day, the Ministry provides a list of countries requiring a visitor visa and a list of countries with medical advisories.

Here are the rules for categorizing travellers.

1. If the required information for an entry record is incomplete, the traveller must be rejected.
2. If any location mentioned in the entry record is unknown, the traveller must be rejected.
3. If the traveller's home country is Kanadia (country code: KAN), the traveller will be accepted.
4. If the reason for entry is to visit and the visitor has a passport from a country from which a visitor visa is required, the traveller must have a valid visa. A valid visa is one that is less than two years old.
5. If the traveller is coming from or travelling through a country with a medical advisory, she or he must be sent to quarantine.

An entry should not be rejected if there is a mismatch between uppercase and lowercase. For example, the case of the country code and passport numbers should not matter.

It is possible for a traveller to receive more than one distinct immigration decisions. Conflicts should be resolved according the order of priority for the immigration decisions: quarantine, reject, and accept.

You may assume that files are in a proper JSON format. (In other words, an improperly formatted file can cause your program to crash.)

You will need to use regular expressions to validate the passport and visa numbers. Each of these should be a function.

In the starter code, a few functions skeletons have been provided to get you started. You should add more functions as necessary.

Testing

Some test data has been provided for you. `countries.json` contains the latest directives from the Ministry. `example_entries.json` is a set of entry records. Be sure to create test data files for your own test cases, consisting of additional json files and even additional test modules.

Evaluation Criteria

Correctness

- Program computes correctly
- Passes test cases provided by initially
- Passes additional test cases added during grading

Design

- Good decisions are made about partitioning functionality
- Appropriate data structures are used
- Modular, no repeated code

Testing

- Positive and negative tests are included
- Cases in the specification are covered
- Includes an adequate number of test cases
- Automated testing used

Style

- Provided function names not changed
- Passes PEP-8 style checker
- Identifiers have meaningful names
- Readability

Documentation

- `docstrings` are provided for the file and functions
- Comments are informative and provide rationale
- Comments do not duplicate code

Process

- All team members take turns pushing to repository
- py.test used
- Code is not added to the repository in a single push
- Commits and pushes are made on a regular basis
- Commit comments are informative

Rubric

Exercise 1		Max Points
	Correctness	8
	Design	4
	Testing	6
	Style	3
	Documentation	3
		/24
Exercise 2		Max Points
	Correctness	16
	Design	16
	Testing	20
	Style	7
	Documentation	7
		/66
Process		/10
Total		/100