# FInal Exam: Sample solutions

Bookmark this page

### problem0 (Score: 10.0 / 10.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 2.0 / 2.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 3.0 / 3.0)

**Important note!** Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

## Problem 0

This problem will expose you to a different but common data format (JSON), the methods that can be used to load data from JSON files, and also tests your data processing/regular expression skills. These can be valuable when you have to collect or scrape your own data in the future.

There are five exercises (numbered 0-4) worth a total of ten (10) points.

### The Data

We are going to play with a mock dataset in the *JavaScript Object Notation (JSON)* format. (You can check out the wiki intro to this format here (https://en.wikipedia.org/wiki/JSON) ).

JSON has it's origins on the web, where it was used as a format to send data from a server to a client, and then handled at the client side using JavaScript (hence the name). It is a common format in data analysis, and is the main format for a great variety of data sources, such as Twitter, Yelp!, Facebook, and many others.

The JSON format is a text format. From your knowledge of Python lists and dictionaries, it should be easy to understand. Let's take a quick look at an example. It's a simple database of individuals with their names, email addresses, gender, location, and lists of friends.

```
[{
    "city": "Sitovo",
    "name": {
        "last_name": "Ricciardo",
        "first_name": "Nerta"
    },
    "id": 1,
    "email": {
        "personal": "nricciardo0@hostgator.com",
        "working": "nricciardo0@java.com"
```

```
        },
        "friends": [
            {
                "last_name": "De'Vere - Hunt",
                "first_name": "Miran"
            },
            {
                "last_name": "Fryers",
                "first_name": "Dorisa"
            },
            {
                "last_name": "Brusin",
                "first_name": "Carina"
            }
        ],
        "gender": "Female"
    }, ...]
```

JSON uses a Python-like syntax to describe values. For instance, the above includes simple values, like strings and integers. It also has collections: square brackets (`[...]`) denote lists of comma-separated elements; while curly brackets (`{...}`) mark dictionaries, which contain comma-separated key-value (or "attribute-value") pairs. And just like Python, these collections may be nested: observe that in some of the key-value pairs, the values are lists or dictionaries.

Indeed, after reading this file into a Python object, you will access data elements using the same 0-based index and dictionary key-value notation. For instance, if that object is stored in a Python variable named `data`, then `data[0]["name"]["last_name"] == "Ricciardo"`.

Run the following code cell, which will make sure you have the data needed for this problem.

In [1]:
```
import requests
import os
import hashlib
import io

def on_vocareum():
    return os.path.exists('.voc')

def on_azure():
    return 'AZURE_NOTEBOOKS_VMVERSION' in os.environ

def on_vuduc_box():
    return os.uname().nodename in ['daffy4.local', 'insomnia']

if on_vocareum():
    DATA_PATH = "../resource/lib/publicdata/json/"
elif on_azure() or on_vuduc_box():
    DATA_PATH = "./"
else:
    print("""
*** Unrecognized platform ***

You will need to manually download a dataset and modify
this code cell to point to it by setting the `DATA_PATH`
variable, below.""")
    DATA_PATH = None  # Path to dataset

assert os.path.exists(DATA_PATH), "Where are the images?"
print("Will look for images having a '.json' extension in '{}'.".format(DATA_PATH))
```

```
Will look for images having a '.json' extension in './'.
```

## Loading JSON in Python

There are several ways in Python to read in a JSON (`.json`) file.

Like **`csv`**, Python has a built-in package called **`json`**, to operate on JSON files. (As always, check out the documentation (https://docs.python.org/3/library/json.html).)

In [2]:
```
import pandas as pd
import json

json_file = open('{}MOCK_DATA.json'.format(DATA_PATH), encoding="utf8")
json_str = json_file.read()
json_data = json.loads(json_str)

# Demo:
print(json_data[0]["name"]["last_name"]) # Should be 'Ricciardo'
```

```
Ricciardo
```

**Exercise 0** (2 points). Complete the following function, **`find_emails(data)`**, so that it returns a to save all the `working` emails in the input file into a list named `emails` and sort it alphabetically in **descending** order.

In [3]:
Student's answer (Top)

```
def find_emails(data):
    ### BEGIN SOLUTION ###
    emails = []
    for items in data:
        emails.append(items['email']['working'])
    emails.sort(reverse=True)
    return emails
    ### END SOLUTION ###
```

In [4]:
Grade cell: `find_emails_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `find_emails`

emails = find_emails(json_data)
assert len(emails) == 1000
assert type(emails) == list
assert emails[0] == 'zwardropege@home.pl'
assert emails[726] == 'dplanfn@newyorker.com'
assert emails[349] == 'mgerbel3j@blogger.com'
assert emails[85] == 'tbenson70@salon.com'
assert emails[899] == 'bdegowe77@unblog.fr'
assert emails[181] == 'rdurbyngz@aboutads.info'
assert emails[703] == 'ebaudq4@sogou.com'
assert emails[156] == 'rsimicqg@gov.uk'
assert emails[483] == 'jsemplenj@google.ru'
assert emails[249] == 'oblitzer30@dropbox.com'
assert emails[134] == 'sbrandomo9@smh.com.au'

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 1** (2 points). Many of the people in tihs dataset have friends. But sadly, some of them don't. :(

Complete the function, **`are_u_lonely(lst)`**, to find the people without friends. The input to the function will be the list created from the JSON file's contents. The function should return a list of names constructed by concatenating the first name and last name together separated by a single space.

(For example, Joanne Goodisson is one of these people. She should be an element in the list shown as `"Joanne Goodisson"`.)

In [5]:
Student's answer (Top)

```
def are_u_lonely(lst):
    ### BEGIN SOLUTION ###
    lonely_people = [p['name']['first_name']+' '+ p['name']['last_name'] for p in lst i
f len(p['friends'])==0]
    return lonely_people
    ### END SOLUTION ###
```

In [6]:

Grade cell: `are_u_lonely_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `are_u_lonely_test`

lonely_guys = are_u_lonely(json_data)
assert len(lonely_guys) == 171, "There are {} lonely guys in your result, shoud be 171"
.format(len(lonely_guys))
assert lonely_guys[2] == 'Joanne Goodisson', 'Joanne should be one of them, but she is
not in your result.'
assert lonely_guys[109] == 'Violetta Swinden', 'Violetta Swinden is missing from your l
ist.'
assert lonely_guys[143] == 'Seumas Turban', 'Seumas Turban is missing from your list.'
assert lonely_guys[78] == 'Giraldo Attard', 'Giraldo Attard is missing from your list.'
assert lonely_guys[127] == 'Gale Ryley', 'Gale Ryley is missing from your list.'
assert lonely_guys[99] == 'Pieter Dillestone', 'Pieter Dillestone is missing from your
list.'
assert lonely_guys[139] == 'Latashia Greenhaugh', 'Latashia Greenhaugh is missing from
your list.'
assert lonely_guys[46] == 'Melodee Malster', 'Melodee Malster is missing from your list
.'
assert lonely_guys[7] == 'Georgine Limprecht', 'Georgine Limprecht is missing from your
list.'
assert lonely_guys[23] == 'Ilse Tackley', 'Ilse Tackley is missing from your list.'
assert lonely_guys[29] == 'Camile Theobalds', 'Camile Theobalds is missing from your li
st.'

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 2** (2 points). Write a function **`edu_users()`**, that finds all users who are using a `'.edu'` domain email address as their personal email, and returns their user ID's as a list.

In [7]:

Student's answer (Top)

```
def edu_users(lst):
    ### BEGIN SOLUTION ###
    return [p['id'] for p in lst if '.edu' in p['email']['personal']]
    ### END SOLUTION ###
```

In [8]:

Grade cell: `edu_users_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `edu_users_test`

test_users = edu_users(json_data)
assert len(test_users) == 53, "You found {} people using edu emails, but there should b
e 53.".format(len(test_users))
assert json_data[test_users[-1]-1]['city'] == 'Alajuela'

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 3** (1 point). Write a function that, given the filename of a JSON file, returns the data as a Pandas dataframe.

Pandas has a convenient function to do just that. Check the documentation for pd.read_json() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_json.html) and use it to complete the function below.

In [9]:

Student's answer (Top)

```
def pd_load_json(file_name):
    ### BEGIN SOLUTION ###
    return pd.read_json(file_name)
```

```
    return pd.read_json(file_name)
    ### END SOLUTION ###

pd_load_json("MOCK_DATA.json").head()
```

Out[9]:

| | city | email | friends | gender | id | name |
|---|---|---|---|---|---|---|
| **0** | Sitovo | {'working': 'nricciardo0@java.com', 'personal'... | [{'first_name': 'Miran', 'last_name': 'De'Vere... | Female | 1 | {'first_name': 'Nerta', 'last_name': 'Ricciardo'} |
| **1** | Jiaoyuan | {'working': 'mlittlewood1@vimeo.com', 'persona... | [{'first_name': 'Dian', 'last_name': 'Hounsham... | Female | 2 | {'first_name': 'Minerva', 'last_name': 'Little... |
| **2** | Zhenchuan | {'working': 'rcasswell2@apple.com', 'personal'... | [{'first_name': 'Meghann', 'last_name': 'Vanna... | Female | 3 | {'first_name': 'Rosa', 'last_name': 'Casswell'} |
| **3** | Rokytne | {'working': 'lbugbee3@odnoklassniki.ru', 'pers... | [] | Female | 4 | {'first_name': 'Loren', 'last_name': 'Bugbee'} |
| **4** | Antipolo | {'working': 'ndenormanville4@tamu.edu', 'perso... | [{'first_name': 'Andrus', 'last_name': 'Szymon... | Female | 5 | {'first_name': 'Neda', 'last_name': 'De Norman... |

In [10]:

Grade cell: `pd_load_json_test` Score: 1.0 / 1.0 (Top)

```
# Test cell: `pd_load_json_test`

test_df = pd_load_json('MOCK_DATA.json')
assert len(test_df) == 1000
assert test_df.columns.tolist() == ['city', 'email', 'friends', 'gender', 'id', 'name']

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 4** (3 points). You should observe that the personal and working email addresses appear in same column? Complete the function, **`split_emails()`** below, so that it separates them into two new columns named `"personal"` and `"working"`. It should return a new dataframe that has all the same columns, but with the `"email"` column removed and replaced by these two new columns. (See the test cell if this is unclear.)

> Hint: There is a nice way of using `.apply` and `pd.Series` to accomplish this task: Stack Overflow (https://stackoverflow.com/questions/38231591/splitting-dictionary-list-inside-a-pandas-column-into-separate-columns) is your friend!

In [11]:

Student's answer (Top)

```
def split_emails(file_name):
    df = pd_load_json(file_name)
    ### BEGIN SOLUTION ###
    new_df = pd.concat([df.drop(['email'], axis=1),
                        df.email.apply(pd.Series)],
                       axis=1)
    return new_df
    ### END SOLUTION ###

split_emails("MOCK_DATA.json").head()
```

Out[11]:

| | city | friends | gender | id | name | personal | working |
|---|---|---|---|---|---|---|---|
| | | [{'first_name': 'Miran', | | | {'first_name': 'Nerta', | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | Sitovo | 'last_name': 'De'Vere... | Female | 1 | 'last_name': 'Ricciardo'} | nricciardo0@hostgator.com | nricciardo0@java.com |
| **1** | Jiaoyuan | [{'first_name': 'Dian', 'last_name': 'Hounsham... | Female | 2 | {'first_name': 'Minerva', 'last_name': 'Little... | mlittlewood1@icio.us | mlittlewood1@vimeo.com |
| **2** | Zhenchuan | [{'first_name': 'Meghann', 'last_name': 'Vanna... | Female | 3 | {'first_name': 'Rosa', 'last_name': 'Casswell'} | rcasswell2@soup.io | rcasswell2@apple.com |
| **3** | Rokytne | [] | Female | 4 | {'first_name': 'Loren', 'last_name': 'Bugbee'} | lbugbee3@dmoz.org | lbugbee3@odnoklassniki.ru |
| **4** | Antipolo | [{'first_name': 'Andrus', 'last_name': 'Szymon... | Female | 5 | {'first_name': 'Neda', 'last_name': 'De Norman... | ndenormanville4@illinois.edu | ndenormanville4@tamu.edu |

In [12]:

Grade cell: `split_emails_test` Score: 3.0 / 3.0 (Top)

```
# Test cell: `Exercise 5`

test_df = split_emails('MOCK_DATA.json')
assert len(test_df) == 1000
assert test_df.columns.tolist() == ['city', 'friends', 'gender', 'id', 'name', 'persona
l', 'working']
assert test_df.personal[0] == 'nricciardo0@hostgator.com'
assert test_df.personal[999] == 'bbretonrr@pen.io'

print("\n(Passed!)")
```

```
(Passed!)
```

**Fin!** You have reached the end of this problem. Don't forget to submit it before moving on.

### problem1 (Score: 10.0 / 10.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 1.0 / 1.0)
6. Test cell (Score: 1.0 / 1.0)
7. Test cell (Score: 1.0 / 1.0)
8. Test cell (Score: 1.0 / 1.0)
9. Test cell (Score: 2.0 / 2.0)

**Important note!** Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

# Problem 1

In this problem, you will write code to "parse" a restricted form of SQL queries. These exercises are about string processing and regular expressions. There are five (5) exercises, numbered 0-4, which are worth a total of ten (10) points.

In [1]:
```
from IPython.display import display

import re
import pandas as pd

# Random number generation for generating test cases:
from random import randrange, randint, choice, sample
```

**Background: SQL review.** Suppose you have two SQL tables, `OneTable` and `AnotherTable`, and you wish to perform an inner-join on that links a column named `ColA` in `OneTable` with `ColB` in `AnotherTable`. Recall that one simple way to do that in SQL would be:

```
SELECT * FROM OneTable, AnotherTable
        WHERE OneTable.ColA = AnotherTable.ColB
```

Or, consider the following more complex example. Suppose you have an additional table named `YetAThird` and you wish to extend the inner-join to include matches between its column, `ColC`, and a second column from `AnotherTable` named `ColB2`:

```
SELECT * FROM OneTable, AnotherTable, YetAThird
        WHERE OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2 = YetAThird.ColC
```

**Exercise 0** (2 points). Suppose you are given a string containing an SQL query in the restricted form,

```
SELECT * FROM [tbls] WHERE [conds]
```

Implement the function, **`split_simple_join(q)`**, so that it takes a query string **`q`** in the form shown above and **returns a pair of substrings** corresponding to `[tbls]` and `[conds]`.

For example, if

```
q == """SELECT * FROM OneTable, AnotherTable, YetAThird
             WHERE OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.ColC"""
```

then

```
split_simple_join(q) == ("OneTable, AnotherTable, YetAThird",
                         "OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.C
olC")
```

**IMPORTANT NOTE!** In this problem, you only need to return the substring between `FROM` and `WHERE` and the one after `WHERE`. You will extract the table names and conditions later on, below.

You should make the following assumptions:

- The input string `q` contains exactly one such query, with no nesting of queries (e.g., no instances of "`SELECT * FROM (SELECT ...)`"). However, the query may (or may not) be a multiline string as shown in the example. (Treat newlines as whitespace.)
- Your function should ignore any leading or trailing whitespace around the SQL keywords, e.g., `SELECT`, `FROM`, and `WHERE`.
- The substring between `SELECT` and `FROM` will be any amount of whitespace, followed by an asterisk (`*`).
- You should **not** treat the SQL keywords in a case-sensitive way; for example, you would regard `SELECT`, `select`, and `sElEct` as the same. However, do **not** change or ignore the case of the non-SQL keywords.
- The `[tbls]` substring contains only a simple list of table names and no other substrings that might be interpreted as SQL keywords.

- The `[conds]` substring contains only table and column names (e.g., `OneTable.ColA`), the equal sign, the `AND` SQL keyword, and whitespace, but no other SQL keywords or symbols.

> Assuming you are using regular expressions for this problem, recall that you can pass `re.VERBOSE` (https://docs.python.org/3/library/re.html#re.VERBOSE) when writing a multiline regex pattern.

In [2]: Student's answer (Top)

```
def split_simple_join(q):
    assert type(q) is str
    ### BEGIN SOLUTION
    match = re.match(r"""\s*[sS][eE][lL][eE][cC][tT]\s+\*
                         \s+[fF][rR][oO][mM]\s+(.*)
                         \s+[wW][hH][eE][rR][eE]\s+(.*)
                         \s*""",
                     q,
                     re.VERBOSE)
    assert match is not None
    return match.groups()[0], match.groups()[1]
    ### END SOLUTION

# Demo
q_demo = """SELECT * FROM OneTable, AnotherTable, YetAThird
              WHERE OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.
ColC"""
print(split_simple_join(q_demo))
```

```
('OneTable, AnotherTable, YetAThird', 'OneTable.ColA = AnotherTable.ColB AND AnotherTable
.ColB2=YetAThird.ColC')
```

In [3]: Grade cell: `split_simple_join_test1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `split_simple_join_test1`

assert split_simple_join(q_demo) == \
           ('OneTable, AnotherTable, YetAThird',
            'OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.ColC')
print("\n(Passed!)")
```

```
(Passed!)
```

In [4]: Grade cell: `split_simple_join_test2` Score: 1.0 / 1.0 (Top)

```
# Test cell: `split_simple_join_test2`

__SQL = {'SELECT', 'FROM', 'WHERE'} # SQL keywords

# Different character classes
__ALPHA = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_'
__ALPHA_VARS = __ALPHA + '_'
__NUM = '0123456789'
__ALPHA_NUM_VARS = __ALPHA_VARS + __NUM
__DOT = '.'
__SPACES = ' \t\n'
__EQ = '='
__ALL = __ALPHA_NUM_VARS + __DOT + __SPACES + __EQ

def flip_coin():
    from random import choice
    return choice([True, False])

def rand_str(k_min, k_max, alphabet):
    """
    Returns a random string of `k` letters chosen uniformly from
    from `alphabet` with replacement, where `k_min <= k <= k_max`
    where `k` is also chosen uniformly at random.
    """
    assert k_max >= k_min >= 0
```

```
    assert k_max >= k_min >= 0
    k = k_min + randint(0, k_max-k_min)
    return ''.join([choice(alphabet) for _ in range(k)])

def rand_spaces(k_max, k_min=0):
    assert k_max >= k_min >= 0
    return rand_str(k_min, k_max, __SPACES)

def rand_case(s):
    """Randomly changes the case of each letter in a string."""
    return ''.join([c.upper() if flip_coin() else c.lower() for c in s])

def rand_var(k_min=1, k_max=10):
    assert k_max >= k_min >= 1
    s = choice(__ALPHA_VARS)
    s += rand_str(k_min-1, k_max-1, __ALPHA_NUM_VARS)
    if s.upper() in __SQL: # Don't generate keywords -- try again
        return rand_var(k_min, k_max)
    return s

def rand_vars(k_min, k_max):
    V = set()
    dk = randint(0, k_max-k_min)
    for k in range(k_min + dk + 1):
        v = rand_var()
        V.add(v)
    return V

def rand_table(max_cols):
    table = rand_var()
    columns = rand_vars(1, max_cols)
    return (table, columns)

def rand_tables(k_min, k_max, c_max=4):
    assert k_max >= k_min >= 1
    num_tables = k_min + randint(0, k_max-k_min)
    tables = {}
    for _ in range(num_tables):
        table, columns = rand_table(c_max)
        while table in tables:
            table, columns = rand_table(v_max, c_max)
        tables = {**tables, **{table: columns}}
    return tables

def rand_field(table_name, col_names):
    return table_name + "." + choice(list(col_names))

def rand_cond(tables):
    assert type(tables) is dict
    assert len(tables) >= 2
    a, b = sample(tables.keys(), 2)
    a_col = rand_field(a, tables[a])
    b_col = rand_field(b, tables[b])
    return (a_col, b_col)

def rand_cond_set(tables, k_min, k_max):
    k = k_min + randint(0, k_max-k_min)
    conds = set()
    while len(conds) < k:
        (a, b) = rand_cond(tables)
        if ((a, b) not in conds) and ((b, a) not in conds):
            conds.add((a, b))
    return conds

def cond_set_to_substrs(conds, max_wspad=4):
    substrs = []
    for a, b in conds:
        s = "{}{}{}{}{}".format(a,
                                rand_spaces(max_wspad),
                                __EQ,
                                rand_spaces(max_wspad),
                                b)
        substrs.append(s)
    return substrs

def substrs_to_str(substrings, sep=',', max_wspad=4):
```

```
    s_final = ''
    for k, s in enumerate(substrings):
        if k > 0:
            s_final += rand_spaces(max_wspad) + sep + rand_spaces(max_wspad)
        s_final += s
    return s_final

def rand_query_ans(max_tables, max_conds, max_wspad=4):
    tables = rand_tables(2, max_tables)
    cond_set = rand_cond_set(tables, 1, 4)
    return tables, cond_set

def pad_1(max_wspad):
    return rand_spaces(max(1, max_wspad), 1)

def form_select(max_wspad=4):
    return pad_1(max_wspad) + rand_case("SELECT") + pad_1(max_wspad) + "*"

def form_from(tables, max_wspad=4):
    from_ans = substrs_to_str(list(tables.keys()), sep=',', max_wspad=max_wspad)
    return pad_1(max_wspad) + rand_case("FROM") + pad_1(max_wspad) + from_ans, from_ans

def form_where(cond_set, max_wspad=4):
    cond_substrs = cond_set_to_substrs(cond_set)
    cond_ans = substrs_to_str(cond_substrs, sep=' AND ', max_wspad=max_wspad)
    return pad_1(max_wspad) + rand_case("WHERE") + pad_1(max_wspad) + cond_ans, cond_an
s

def form_query_str(tables, cond_set, max_wspad=4):
    select_clause = form_select(max_wspad)
    from_clause, from_ans = form_from(tables, max_wspad)
    where_clause, cond_ans = form_where(cond_set, max_wspad)
    query = select_clause + from_clause + where_clause
    return query, from_ans, cond_ans

def split_simple_join_battery(num_tests, max_wspad=4):
    for k in range(num_tests):
        tables, cond_set = rand_query_ans(5, 5, max_wspad)
        qstmt, from_ans, where_ans = form_query_str(tables, cond_set, max_wspad)
        print("=== Test Statement {} ===\n'''{}'''\n".format(k, qstmt))
        print("True 'FROM' clause substring: '''{}'''\n".format(from_ans))
        print("True 'WHERE' clause substring: '''{}'''\n".format(where_ans))

split_simple_join_battery(5, 3)

print("\n(Passed!)")
```

```
=== Test Statement 0 ===
'''
seLEct

*
 frOm
XH
 ,
RBJHpjPOyH,
        t8bDR2           ,
        ypyANUvVv
 wHere
XH.Dkwh =
 RBJHpjPOyH.V4gpJYmTAp'''

True 'FROM' clause substring: '''XH
 ,
RBJHpjPOyH,
        t8bDR2           ,
        ypyANUvVv'''

True 'WHERE' clause substring: '''XH.Dkwh       =
 RBJHpjPOyH.V4gpJYmTAp'''

=== Test Statement 1 ===
'''     SeLEcT  *
        from     Y7
```

```
,       jnT
,               F8dDmJUmG,y                     WHEre
       F8dDmJUmG.oHG
 =
Y7.zW'''

True 'FROM' clause substring: '''Y7
,       jnT
,               F8dDmJUmG,y'''

True 'WHERE' clause substring: '''F8dDmJUmG.oHG
 =
Y7.zW'''

=== Test Statement 2 ===
'''
SeLECt  *

fROm

J1o7DPfv,         tfAxjg,J7FnS WHeRe

tfAxjg.UFGyZqLm

=
               J7FnS._IWJ

 AND
 J1o7DPfv.qy0Kg5oqG_

=

 J7FnS.SB        AND
       J1o7DPfv.q9jR
 = tfAxjg.q0kZVI         AND    J7FnS.KhphewZ = tfAxjg.H'''

True 'FROM' clause substring: '''J1o7DPfv,        tfAxjg,J7FnS'''

True 'WHERE' clause substring: '''tfAxjg.UFGyZqLm

=
               J7FnS._IWJ

 AND
 J1o7DPfv.qy0Kg5oqG_

=

 J7FnS.SB        AND
       J1o7DPfv.q9jR
 = tfAxjg.q0kZVI         AND    J7FnS.KhphewZ = tfAxjg.H'''

=== Test Statement 3 ===
'''
               SELeCT   *
froM
LiVYRHcx ,      U               ,uAae1Zepk
whERE
uAae1Zepk.mA            =
U.iFGBDOroYs            AND    uAae1Zepk.NU
=  U.r0LRqmWmP  AND  uAae1Zepk.HLMVvo1Io= LiVYRHcx.UCCxhIu6
 AND

U.iFGBDOroYs
  =

LiVYRHcx.f1RycFfl99'''

True 'FROM' clause substring: '''LiVYRHcx ,     U               ,uAae1Zepk'''

True 'WHERE' clause substring: '''uAae1Zepk.mA          =
U.iFGBDOroYs            AND    uAae1Zepk.NU
=  U.r0LRqmWmP  AND  uAae1Zepk.HLMVvo1Io= LiVYRHcx.UCCxhIu6
 AND
```

```
U.iFGBDOroYs
  =

LiVYRHcx.f1RycFfl99'''

=== Test Statement 4 ===
'''
seLECt
 *      FrOM
w ,Jwq4H                whErE
w.M
=       Jwq4H.q  AND            Jwq4H.RDWfhc            =

w.Yx8Py  AND w.lWk=Jwq4H.dIk9'''

True 'FROM' clause substring: '''w ,Jwq4H'''

True 'WHERE' clause substring: '''w.M
=       Jwq4H.q  AND            Jwq4H.RDWfhc            =

w.Yx8Py  AND w.lWk=Jwq4H.dIk9'''


(Passed!)
```

**Variable names.** For this problem, let a valid *variable name* be a sequence of alphanumeric characters or underscores, where the very first character *cannot* be a number. For example,

```
some_variable
__another_VariAble
Yet_a_3rd_var
_A_CSE_6040_inspired_var
```

are all valid variable names, whereas the following are not.

```
123var_is_bad
0_is_not_good_either
4goodnessSakeStopItAlready
```

**Exercise 1** (2 points). Implement a function, **`is_var(s)`**, that checks whether a valid variable name. That is, it should return `True` if and only if `s` is valid according to the above definition. Your function should ignore any leading or trailing spaces in `s`.

For example:

```
assert is_var("__another_VariAble")
    assert not is_var("0_is_not_good_either")
    assert is_var("   foo")
    assert is_var("_A_CSE_6040_inspired_var   ")
    assert not is_var("#getMe2")
    assert is_var("   Yet_a_3rd_var  ")
    assert not is_var("123var_is_bad")
    assert not is_var("  A.okay")
```

In [5]: Student's answer (Top)

```
def is_var(s):
    assert type(s) is str
    ### BEGIN SOLUTION
    matches = re.match("""^\s*        # Leading spaces
                       (
                        [a-zA-Z_]  # Start of variable
                        \w*        # Remaining...
                       )
                       \s*$        # Trailing spaces
                      """, s, re.VERBOSE)
    return matches is not None
```

```
    ### END SOLUTION
```

In [6]:

Grade cell: `is_var_test0` Score: 1.0 / 1.0 (Top)

```
# Test cell, part 1: `is_var_test0`

assert is_var("__another_VariAble")
assert not is_var("0_is_not_good_either")
assert is_var("   foo")
assert is_var("_A_CSE_6040_inspired_var   ")
assert not is_var("#getMe2")
assert is_var("   Yet_a_3rd_var  ")
assert not is_var("123var_is_bad")
assert not is_var("  A.okay")

print("\n(Passed part 1 of 2.)")
```

```
(Passed part 1 of 2.)
```

In [7]:

Grade cell: `is_var_test1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `is_var_test2`

for v in rand_vars(20, 30):
    ans = flip_coin()
    if not ans:
        v = choice(__NUM) + v
    v = rand_spaces(3) + v + rand_spaces(3)
    your_ans = is_var(v)
    assert your_ans == ans, "is_var('{}') == {} instead of {}.".format(v, your_ans, ans
)

print("\n(Passed part 2 of 2.)")
```

```
(Passed part 2 of 2.)
```

**Column variables.** A *column variable* consists of two valid variable names separated by a single period. For example,

```
A.okay
a32X844._387b
__C__.B3am
```

are all examples of column variables: in each case, the substrings to the left and right of the period are valid variables.

**Exercise 2** (1 point). Implement a function, **`is_col(s)`**, so that it returns `True` if and only if **`s`** is a column variable, per the definition above.

For example:

```
assert is_col("A.okay")
    assert is_col("a32X844._387b")
    assert is_col("__C__.B3am")
    assert not is_col("123.abc")
    assert not is_col("abc.123")
```

As with Exercise 1, your function should ignore any leading or trailing spaces.

In [8]:

Student's answer (Top)

```
def is_col(s):
    ### BEGIN SOLUTION
    matches = re.match("""^\s*        # Leading spaces
                          (
                           [a-zA-Z_]  # Start of variable
                           \w*        # Remaining
```

```
                            \w*         # Remaining...
                           )
                           \.          # Separator
                           (
                            [a-zA-Z_]  # Start of variable
                            \w*        # Remaining...
                           )
                           \s*$        # Trailing spaces
                          """, s, re.VERBOSE)
    return matches is not None
    ### END SOLUTION
```

In [9]:

Grade cell: `is_col_test0` Score: 1.0 / 1.0 (Top)

```
# Test cell: `is_col_test0`

assert is_col("A.okay")
assert is_col("a32X844._387b")
assert is_col("__C__.B3am")
assert not is_col("123.abc")
assert not is_col("abc.123")

print("\n(Passed part 1.)")
```

```
(Passed part 1.)
```

In [10]:

Grade cell: `is_col_test1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `is_col_test1`

def test_is_col_1():
    a = rand_var()
    assert not is_col(a), "is_col('{}') == {} instead of {}.".format(a, is_col(a), Fals
e)
    a_valid = flip_coin()
    if not a_valid:
        a = rand_str(1, 5, __NUM)
    return a, a_valid

for _ in range(20):
    a, a_valid = test_is_col_1()
    b, b_valid = test_is_col_1()
    ans = a_valid and b_valid

    c = "{}{}.{}{}".format(rand_spaces(3), a, b, rand_spaces(3))
    your_ans = is_col(c)
    print("==> is_col('{}') == {}".format(c, your_ans))
    assert your_ans == ans

print("\n(Passed part 2.)")
```

```
==> is_col('
GBkgaK.DMWJY
') == True
==> is_col('  73349.948          ') == False
==> is_col('
RzNyoQB.063                     ') == False
==> is_col('805.IL4ZixTc2
') == False
==> is_col('    4.406  ') == False
==> is_col('

 78245.69546                    ') == False
==> is_col('   ldIC.affzt
        ') == True
==> is_col('
k.712
 ') == False
==> is_col('551.Y7oxKAdV0G') == False
==> is_col('
```

```
        WyoK4a.i
        ') == True
==> is_col('  735.m9ud

') == False
==> is_col('2293.CIO25O ') == False
==> is_col('Sc9tpd.O


') == True
==> is_col('    nAfh.1  ') == False
==> is_col('
        I77JP7b.3779') == False
==> is_col('YkWLXlDwCt.78026 ') == False
==> is_col('

 UACU.n78               ') == True
==> is_col(' 5.Y        ') == False
==> is_col('


99.XQ4O ') == False
==> is_col('            757.2 ') == False

(Passed part 2.)
```

**Equality strings.** An *equality string* is a string of the form,

```
A.x = B.y
```

where `A.x` and `B.y` are *column variable* names and = is an equals sign. There may be any amount of whitespace---including none---before or after each variable and the equals sign.

**Exercise 3** (2 points). Implement the function, **`extract_eqcols(s)`**, below. Given an input string **`s`**, if it is an equality string, your function should return a pair `(u, v)`, where `u` and `v` are the two column variables in the equality string. For example:

```
assert extract_eqcols("F3b._xyz =AB0_.def") == ("F3b._xyz", "AB0_.def")
```

If `s` is not a valid equality string, then your function should return `None`.

In [11]:

Student's answer (Top)

```
def extract_eqcols(s):
    ### BEGIN SOLUTION
    parts = s.split('=')
    if len(parts) == 2 and is_col(parts[0]) and is_col(parts[1]):
        return (parts[0].strip(), parts[1].strip())
    return None
    ### END SOLUTION

print(extract_eqcols("F3b._xyz =AB0_.def"))
```

```
('F3b._xyz', 'AB0_.def')
```

In [12]:

Grade cell: `extract_eqcols0` Score: 1.0 / 1.0 (Top)

```
# Test cell: `extract_eqcols0`

assert extract_eqcols("F3b._xyz =AB0_.def") == ("F3b._xyz", "AB0_.def")
assert extract_eqcols("0F3b._xyz =AB0_.def") is None

print("\n(Passed part 1 of 2.)")
```

```
(Passed part 1 of 2.)
```

In [13]:

In [13]: Grade cell: `extract_eqcols1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `extract_eqcols1`

for _ in range(5):
    _, cond_set = rand_query_ans(2, 10, 5)
    for a, b in cond_set:
        s = a + rand_spaces(3) + __EQ + rand_spaces(3) + b
        print("==> Processing:\n'''{}'''\n".format(s))
        ans = extract_eqcols(s)
        print("    *** Found: {} ***".format(ans))
        assert ans is not None, "Did not detect an equality string where there was one!
"
        assert ans[0] == a and ans[1] == b, "Returned {} instead of ({}, {})".format(an
s, a, b)

print("\n(Passed part 2 of 2.)")
```

```
==> Processing:
'''R.MSOfBi1Lk=Phw7.i2rhy6v5'''

    *** Found: ('R.MSOfBi1Lk', 'Phw7.i2rhy6v5') ***
==> Processing:
'''yFNmd1.eiAXYa=        bC9IZYV.BiTByJ6Y'''

    *** Found: ('yFNmd1.eiAXYa', 'bC9IZYV.BiTByJ6Y') ***
==> Processing:
'''yFNmd1.eiAXYa

=                       bC9IZYV.d_Z'''

    *** Found: ('yFNmd1.eiAXYa', 'bC9IZYV.d_Z') ***
==> Processing:
'''ZEon.w        =
        poMCf.Kwp3Kt7_B'''

    *** Found: ('ZEon.w', 'poMCf.Kwp3Kt7_B') ***
==> Processing:
'''poMCf.HnjvC

=       ZEon.ihUbD'''

    *** Found: ('poMCf.HnjvC', 'ZEon.ihUbD') ***
==> Processing:
'''T4.sW                =               ACNVaas.DmmkCquz_'''

    *** Found: ('T4.sW', 'ACNVaas.DmmkCquz_') ***
==> Processing:
'''T4.P          =
ACNVaas.CAX6m'''

    *** Found: ('T4.P', 'ACNVaas.CAX6m') ***
==> Processing:
'''ACNVaas.U    =T4.sW'''

    *** Found: ('ACNVaas.U', 'T4.sW') ***
==> Processing:
'''w9EA2.mzTS                   =
 GErJACBrNO._iJZyI'''

    *** Found: ('w9EA2.mzTS', 'GErJACBrNO._iJZyI') ***
==> Processing:
'''GErJACBrNO._iJZyI     =

w9EA2.c_meTLtBJ'''

    *** Found: ('GErJACBrNO._iJZyI', 'w9EA2.c_meTLtBJ') ***
==> Processing:
'''w9EA2.mzTS   =GErJACBrNO.ZlY_'''

    *** Found: ('w9EA2.mzTS', 'GErJACBrNO.ZlY_') ***
==> Processing:
'''w9EA2.VpJWyPscgf=
        GErJACBrNO._iJZyI'''
```

```
    *** Found: ('w9EA2.VpJWyPscgf', 'GErJACBrNO._iJZyI') ***

(Passed part 2 of 2.)
```

**Exercise 4** (2 points). Given an SQL query in the restricted form described above, write a function that extracts all of the join conditions from the `WHERE` clause. Name this fuction, **`extract_join_conds(q)`**, where `q` is the query string. It should return a list of pairs, where each pair `(a, b)` is the name of the left- and right-hand sides in one of these conditions.

For example, suppose:

```
q == """SELECT * FROM OneTable, AnotherTable, YetAThird
            WHERE OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.ColC"""
```

Notice that the `WHERE` clause contains two conditions: `OneTable.ColA = AnotherTable.ColB` and `AnotherTable.ColB2=YetAThird.ColC`. Therefore, your function should return a list of two pairs, as follows:

```
extract_join_conds(q) == [("OneTable.ColA", "AnotherTable.ColB"),
                          ("AnotherTable.ColB2", "YetAThird.ColC")]
```

In [14]: Student's answer (Top)

```
def extract_join_conds(q):
    ### BEGIN SOLUTION
    _, where_clause = split_simple_join(q)
    conds_str = re.split("\s+[aA][nN][dD]\s+", where_clause)
    return [extract_eqcols(c) for c in conds_str]
    ### END SOLUTION

print("==> Query:\n\t'{}'\n".format(q_demo))
print("==> Results:\n{}".format(extract_join_conds(q_demo)))
```

```
==> Query:
        'SELECT * FROM OneTable, AnotherTable, YetAThird
              WHERE OneTable.ColA = AnotherTable.ColB AND AnotherTable.ColB2=YetAThird.Co
lC'

==> Results:
[('OneTable.ColA', 'AnotherTable.ColB'), ('AnotherTable.ColB2', 'YetAThird.ColC')]
```

In [15]: Grade cell: `extract_join_conds_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `extract_join_conds_test`

def test_extract_join_conds_1():
    tables, cond_set = rand_query_ans(5, 5, 0)
    qstmt, _, _ = form_query_str(tables, cond_set, 0)
    qstmt = re.sub("[\n\t]", " ", qstmt)
    print("=== {} ===\n".format(qstmt))
    print("  True solution: {}\n".format(cond_set))
    your_conds = extract_join_conds(qstmt)
    print("  Your solution: {}\n".format(your_conds))
    assert set(your_conds) == cond_set, "*** Mismatch? ***"

for _ in range(10):
    test_extract_join_conds_1()

print("\n(Passed!)")
```

```
===  SELect * FRom qDsSfyFV_0,xFfHAovgQT,anINTz3tYA,SZsLduRbKn wHeRe qDsSfyFV_0.oL2zT  =a
nINTz3tYA.MvlG1dZ AND qDsSfyFV_0.oL2zT=anINTz3tYA.QhFj AND qDsSfyFV_0.eyZ16OU = anINTz3tY
A.MvlG1dZ AND qDsSfyFV_0.oL2zT= SZsLduRbKn.PD6Sk ===

  True solution: {('qDsSfyFV_0.oL2zT', 'anINTz3tYA.MvlG1dZ'), ('qDsSfyFV_0.oL2zT', 'anINT
z3tYA.QhFj'), ('qDsSfyFV_0.eyZ16OU', 'anINTz3tYA.MvlG1dZ'), ('qDsSfyFV_0.oL2zT', 'SZsLduR
bKn.PD6Sk')}

  Your solution: [('qDsSfyFV_0.oL2zT', 'anINTz3tYA.MvlG1dZ'), ('qDsSfyFV_0.oL2zT', 'anINT
```

```
z3tYA.QhFj'), ('qDsSfyFV_0.eyZ16OU', 'anINTz3tYA.MvlG1dZ'), ('qDsSfyFV_0.oL2zT', 'SZsLduR
bKn.PD6Sk')]

===  seLeCt * fRoM wtv_rQS1ke,qSIemufA,WV,N WherE qSIemufA.HMcc5iE8   =  N.ppnp828MWm AND
WV.CW2iq    =   N.ppnp828MWm ===

  True solution: {('qSIemufA.HMcc5iE8', 'N.ppnp828MWm'), ('WV.CW2iq', 'N.ppnp828MWm')}

  Your solution: [('qSIemufA.HMcc5iE8', 'N.ppnp828MWm'), ('WV.CW2iq', 'N.ppnp828MWm')]

===  sELECt * fROm kMg8n,a1B5z3,r_JwSX48Yk,NVWVq wHErE kMg8n.XDWWG    =NVWVq._zGIfs8 AND
NVWVq.O=    a1B5z3.Q1Ol ===

  True solution: {('kMg8n.XDWWG', 'NVWVq._zGIfs8'), ('NVWVq.O', 'a1B5z3.Q1Ol')}

  Your solution: [('kMg8n.XDWWG', 'NVWVq._zGIfs8'), ('NVWVq.O', 'a1B5z3.Q1Ol')]

===  SEleCt * From pawc,Diej6eSKL,oM9sA,x wHERE pawc.CJucWmtxz_    =  oM9sA.YtWee ===

  True solution: {('pawc.CJucWmtxz_', 'oM9sA.YtWee')}

  Your solution: [('pawc.CJucWmtxz_', 'oM9sA.YtWee')]

===  SelEcT * FRom OzlpO3u,VzrDppX wHerE OzlpO3u.rgJnTDN   =    VzrDppX.VMg AND OzlpO3u.r
gJnTDN   =  VzrDppX.UGo ===

  True solution: {('OzlpO3u.rgJnTDN', 'VzrDppX.VMg'), ('OzlpO3u.rgJnTDN', 'VzrDppX.UGo')}

  Your solution: [('OzlpO3u.rgJnTDN', 'VzrDppX.VMg'), ('OzlpO3u.rgJnTDN', 'VzrDppX.UGo')]

===  selEct * frOM ipWt,KiueLl1hY,hw,lCP wHerE hw.OnOK  =  ipWt.V5ZI ===

  True solution: {('hw.OnOK', 'ipWt.V5ZI')}

  Your solution: [('hw.OnOK', 'ipWt.V5ZI')]

===  SEleCT * fROm f0Gbhy,Aw WheRe f0Gbhy.OC1mmpSuu=  Aw.z_WxBm AND Aw.vmluO7 =    f0Gbhy
.dJK8wPbLzt AND Aw.z_WxBm    =   f0Gbhy.dJK8wPbLzt AND f0Gbhy.OC1mmpSuu = Aw.vmluO7 ===

  True solution: {('f0Gbhy.OC1mmpSuu', 'Aw.z_WxBm'), ('Aw.vmluO7', 'f0Gbhy.dJK8wPbLzt'),
('Aw.z_WxBm', 'f0Gbhy.dJK8wPbLzt'), ('f0Gbhy.OC1mmpSuu', 'Aw.vmluO7')}

  Your solution: [('f0Gbhy.OC1mmpSuu', 'Aw.z_WxBm'), ('Aw.vmluO7', 'f0Gbhy.dJK8wPbLzt'),
('Aw.z_WxBm', 'f0Gbhy.dJK8wPbLzt'), ('f0Gbhy.OC1mmpSuu', 'Aw.vmluO7')]

===  selecT * FRoM o0x6ip6k,Q1H,mmbyTw_O WhERE Q1H.t1N3  =   o0x6ip6k.InDuJHsF ===

  True solution: {('Q1H.t1N3', 'o0x6ip6k.InDuJHsF')}

  Your solution: [('Q1H.t1N3', 'o0x6ip6k.InDuJHsF')]

===  sEleCT * FRoM r10AkaK59,sf,iWWToyVoY,g WHeRe iWWToyVoY.NuCz1 =sf.lzV0VYn ===

  True solution: {('iWWToyVoY.NuCz1', 'sf.lzV0VYn')}

  Your solution: [('iWWToyVoY.NuCz1', 'sf.lzV0VYn')]

===  sELecT * FRoM UIrsoW,qRFJrDrp,MWVSHmJDJ,h,awB1wFN WHere UIrsoW.Wx    =MWVSHmJDJ.gf9q
oHt8GW AND MWVSHmJDJ.sRDr=  UIrsoW.Wx AND MWVSHmJDJ.sRDr =   h.CR1 ===

  True solution: {('UIrsoW.Wx', 'MWVSHmJDJ.gf9qoHt8GW'), ('MWVSHmJDJ.sRDr', 'UIrsoW.Wx'),
('MWVSHmJDJ.sRDr', 'h.CR1')}

  Your solution: [('UIrsoW.Wx', 'MWVSHmJDJ.gf9qoHt8GW'), ('MWVSHmJDJ.sRDr', 'UIrsoW.Wx'),
('MWVSHmJDJ.sRDr', 'h.CR1')]


(Passed!)
```

**Fin!** This marks the end of this problem. Don't forget to submit it to get credit.

problem2 (Score: 10.0 / 10.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 3.0 / 3.0)
3. Test cell (Score: 2.0 / 2.0)
4. Test cell (Score: 3.0 / 3.0)

**Important note!** Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

## Problem 2

Millions of searches happen on modern search engines like Google. Advertisers want to know about search interests in order to target consumers effectively. In this notebook, we will look at "search interest scores" for the 2016 Olympics obtained from Google Trends (https://trends.google.com/trends/).

This problem is divided into four (4) exercises, numbered 0-3. They are worth a total of ten (10) points.

> By way of background, a search interest score is computed by region and normalized by population size, in order to account for differences in populations between different regions. You can read more about search interest here. https://medium.com/google-news-lab/what-is-google-trends-data-and-what-does-it-mean-b48f07342ee8 (https://medium.com/google-news-lab/what-is-google-trends-data-and-what-does-it-mean-b48f07342ee8)

In [1]:
```
# Some modules and functions we'll need

import pandas as pd
import sqlite3
from IPython.display import display

def canonicalize_tibble(X):
    """Returns a tibble in _canonical order_."""
    # Enforce Property 1:
    var_names = sorted(X.columns)
    Y = X[var_names].copy()
```

```
    # Enforce Property 2:
    Y.sort_values(by=var_names, inplace=True)

    # Enforce Property 3:
    Y.set_index([list(range(0, len(Y)))], inplace=True)

    return Y

def tibbles_are_equivalent(A, B):
    """Given two tidy tables ('tibbles'), returns True iff they are
    equivalent.
    """
    A_canonical = canonicalize_tibble(A)
    B_canonical = canonicalize_tibble(B)
    cmp = A_canonical.eq(B_canonical)
    return cmp.all().all()
```

## The data

We will be working with two sources of data.

The first is the search interest data taken from Google Trends (https://raw.githubusercontent.com/googletrends/data/master/20160819_OlympicSportsByCountries.csv).

The second is world population data taken from the U.S. Census Bureau (https://www.census.gov/population/international/data/idb/).

For your convenience, these data are stored in two tables in a SQLite database stored in a file named `olympics/sports.db`. We will need to read the data into dataframes before proceeding.

**Exercise 0** (2 points). The SQLite database has two tables in it, one named `search_interest` and the other named `countries`. Implement the function, **`read_data(conn)`** below, to read these tables into a pair of Pandas dataframes.

In particular, assume that **`conn`** is an open SQLite database connection object. Your function should return a pair of dataframes, `(search_interest, countries)`, corresponding to these tables. (See the `# Demo code` below.)

In [2]:

Student's answer (Top)

```
def read_data(conn):
    ### BEGIN SOLUTION
    df1 = pd.read_sql('select * from search_interest', conn)
    df2 = pd.read_sql('select * from countries', conn)
    return df1, df2
    ### END SOLUTION

# Demo code:
conn = sqlite3.connect('olympics/sports.db')
search_interest, countries = read_data(conn)
conn.close()

print("=== search_interest ===")
display(search_interest.head())

print("=== countries ===")
display(countries.head())
```

```
=== search_interest ===
```

| | index | Country | Search_Interest | Sport |
|---|---|---|---|---|
| **0** | 0 | Iran | 1 | Archery |
| **1** | 1 | South Korea | 2 | Archery |
| **2** | 2 | Mexico | 1 | Archery |
| **3** | 3 | Netherlands | 1 | Archery |
| **4** | 4 | Aruba | 16 | Artistic gymnastics |

```
=== countries ===
```

| | index | Country | Year | Population | Area_sq_km | Density |
|---|---|---|---|---|---|---|
| **0** | 0 | Reunion | 2016 | 850996 | 2511 | 340.0 |
| **1** | 1 | Martinique | 2016 | 385551 | 128 | 340.0 |
| **2** | 2 | Guadeloupe | 2016 | 402119 | 1628 | 250.0 |
| **3** | 3 | Myanmar | 2016 | 54616716 | 653508 | 83.6 |
| **4** | 4 | CzechRepublic | 2016 | 10660932 | 77247 | 138.0 |

In [3]:

Grade cell: `read_data_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `read_data_test`

df1 = pd.read_csv("olympics/OlympicSportsByCountries_2016.csv")
df2 = pd.read_csv("olympics/census_data_2016.csv")

try:
    ref = pd.read_csv
    del pd.read_csv
    conn = sqlite3.connect('olympics/sports.db')
    search_interest, countries = read_data(conn)
    conn.close()
except AttributeError as e:
    raise RuntimeError("Were you using read_csv to read the csv solution ?")
finally:
    pd.read_csv = ref

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 1** (3 points). In this exercise, compute the answers to the following three questions about the `search_interests` data.

1. Which country has the "most varied" interest in Olympic sports? That is, in the dataframe of search interests, which country appears most often? Store the result in the variable named **`top_country`**.
2. Which Olympic sport generates interest in the largest number of countries? Store the result in the variable **`top_sport`**.
3. How many sports are listed in the table? Store the result in the variable **`sport_count`**.

> **Hint** : The `scipy.stats.mode()` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html) function could be useful in this exercise.

In [4]:

Student's answer (Top)

```
from scipy.stats import mode

top_country = None
top_sport = None
sport_count = None

def compute_basic_stats():
    ### BEGIN SOLUTION ###
    top_country = mode(search_interest["Country"].tolist())[0]
    top_sport = mode(search_interest["Sport"].tolist())[0]
    sport_count = len(search_interest["Sport"].unique())
    ### END SOLUTION ###

    return top_country, top_sport, sport_count

top_country, top_sport, sport_count = compute_basic_stats()
```

In [5]:

Grade cell: `compute_basic_stats_test` Score: 3.0 / 3.0 (Top)

```
# Test code
```

```
# test code
try:
    ref = search_interest
    del search_interest
    top_country, top_sport, sport_count = compute_basic_stats()
except NameError:
    search_interest = ref
    top_country, top_sport, sport_count = compute_basic_stats()
    assert top_country == 'Croatia' or top_country == 'New Zealand'
    assert top_sport == 'Athletics (Track & Field)'
    assert sport_count == 34
except Exception as e:
    print(e)
    print("Were you not using the search_interest dataframe to compute the stats ?")
finally:
    search_interest = ref

print("\n(Passed!)")
```

```
(Passed!)
```

## Worldwide popularity of a sport

To estimate the popularity of a sport, it is not good enough to get only a count of the countries where the sport generated enough search interest. We might get a better estimate of popularity by computing a weighted average of search interest that accounts for differences in search interests and populations among countries.

**Exercise 2** (2 points). Before we can perform a weighted average, we need to find the weights for each country. To do that, we need the population for each of the countries in the search interest table, which we can obtain by querying the census population table.

Complete the function **`join_pop(si, c)`** below to perform this task. That is, given the dataframe of search interests, **`si`**, and the census data, **`c`**, this function should join the `Population` column from `c` to `si` and return the result.

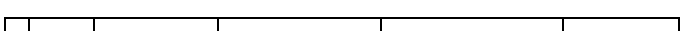
The returned value of `join_pop(si, c)` should be a copy of `si` with one additional column named `'Population'` that holds the corresponding population value from `c`.

> To match the country names between the `si` and `c` dataframes, note that the `si` dataframe's `'Country'` column includes spaces whereas `c` does not. You'll want to account for that by, for instance, stripping out the spaces from `si` before merging or joining with `c`.

In [6]:

Student's answer (Top)

```
def translate_country(country):
    """
    Removes spaces from country names
    """
    return country.replace(' ', '')

def join_pop(si, c):
    join_df = si.copy()
    ### BEGIN SOLUTION ###
    join_df['foreign_key'] = si["Country"].apply(translate_country)
    join_df = pd.merge(join_df, c[["Country", "Population"]],
                       left_on='foreign_key', right_on='Country',
                       how='left',
                       suffixes=('', '_y'))
    join_df.drop(['foreign_key', 'Country_y'], axis=1, inplace=True)
    ### END SOLUTION ###
    return join_df

total_world_population = sum(countries["Population"])
join_df = join_pop(search_interest, countries)

display(join_df.head())
```



| | index | Country | Search_Interest | Sport | Population |
|---|---|---|---|---|---|
| **0** | 0 | Iran | 1 | Archery | 80987449 |
| **1** | 1 | South Korea | 2 | Archery | 50924172 |
| **2** | 2 | Mexico | 1 | Archery | 123166749 |
| **3** | 3 | Netherlands | 1 | Archery | 17016967 |
| **4** | 4 | Aruba | 16 | Artistic gymnastics | 113648 |

In [7]:

Grade cell: `join_tables_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `join_tables_test`

join_df_ref = pd.read_csv("olympics/joined_df.csv")

try:
    ref = pd.read_csv
    del pd.read_csv
    join_df = join_pop(search_interest, countries)
    assert tibbles_are_equivalent(join_df, join_df_ref), "Solution is incorrect"
except AttributeError as e:
    raise RuntimeError("Were you using read_csv to read the csv solution ?")
finally:
    pd.read_csv = ref

print("\n(Passed!)")
```

```
(Passed!)
```

**Weighing search interest by population.** Suppose that to compare different Olympic sports by global popularity, we want to account for each country's population.

For instance, suppose we are looking at the global search interest in volleyball. If volleyball's search interest equals `1` in both China and the Netherlands, we might weigh China's search interest more since it is the more populous contry.

To determine the weights for each country, let's just use each country's fraction of the global population. Recall that an earlier code cell computed the variable, `total_world_population`, which is the global population. Let the weight of a given country be its population divided by the global population. (For instance, if the global population is 6 billion people and the population of India is 1 billion, then India's "weight" would be one-sixth.)

**Exercise 3** (3 points). Create a dataframe named `ranking` with two columns, `'Sport'` and `'weighted_interest'`, where there is one row per sport, and each sport's `'weighted_interest'` is the overall weighted interest across countries, using the population weights for each country as described above.

> **Hint**: Consider using groupby() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html) for Pandas DataFrames. It is very similar to `GROUP BY` in SQL.

In [8]:

Student's answer (Top)

```
### BEGIN SOLUTION ###
join_df["weighted_interest"] = join_df["Population"] * join_df["Search_Interest"] / tot
al_world_population
ranking = join_df[["Sport", "weighted_interest"]]
ranking = ranking.groupby("Sport").sum()
ranking.sort_values(by="weighted_interest", ascending=False, inplace=True)
ranking.reset_index(inplace=True)
### END SOLUTION ###

# top 10 sports
display(ranking[:10])
```

| | Sport | weighted_interest |
|---|---|---|

| | | |
|---|---|---|
| **0** | Swimming | 5.983388 |
| **1** | Athletics (Track & Field) | 4.273728 |
| **2** | Badminton | 3.051064 |
| **3** | Artistic gymnastics | 2.337363 |
| **4** | Tennis | 2.119308 |
| **5** | Football (Soccer) | 1.345433 |
| **6** | Table tennis | 0.929301 |
| **7** | Wrestling | 0.845934 |
| **8** | Diving | 0.727840 |
| **9** | Basketball | 0.462788 |

In [9]:

Grade cell: `ranking_test` Score: 3.0 / 3.0 (Top)

```
# Test cell: `ranking_test`

ranking_ref = pd.read_csv("olympics/rankings_ref.csv")
assert (ranking_ref["Sport"] == ranking["Sport"]).all()

print("\n(Passed!)")
```

```
(Passed!)
```

**Fin!** You have reached the end of this problem. Be sure to submit it before moving on.

---

### problem3 (Score: 10.0 / 10.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 2.0 / 2.0)
4. Test cell (Score: 2.0 / 2.0)
5. Test cell (Score: 1.0 / 1.0)
6. Test cell (Score: 1.0 / 1.0)
7. Test cell (Score: 1.0 / 1.0)

**Important note!** Before you turn in this lab notebook, make sure everything runs as expected:

- First, **restart the kernel** -- in the menubar, select Kernel→Restart.
- Then **run all cells** -- in the menubar, select Cell→Run All.

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE."

# Problem 3

This problem is about counting triangles in a graph, which has applications in social network analysis.

It has **four (4)** exercises, numbered 0-3, which are worth a total of ten (10) points.

## Background: Counting triangles in a social network

A social network may be modeled as an undirected graph, like the one shown below.

The *nodes* (or *vertices*) of this graph, shown as numbered circles, might represent people, and the *edges* (or links connecting them) might represent who is friends with whom. In this case, person 0 is friends with all the "odd birds" of this network, persons 1, 3, and 5, but has no connection to persons 2 and 4.

**Adjacency matrix.** Let $A$ be the *adjacency matrix* representation of the graph, defined as follows. The entries of $A$ are either 0 or 1; and $a_{i,j}$ equals 1 if and only if there is an edge connecting nodes $i$ and $j$. For instance, for the graph shown above,

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

Observe that the relationships are symmetric. For instance, 0 and 1 are mutually connected; therefore, both $a_{0,1}$ and $a_{1,0}$ equal 1, and in general, $A = A^T$.

**Counting triangles.** One type of analysis one might perform on such a graph is *counting triangles*, that is, the number of relationships of the form $a$ knows $b$, $b$ knows $c$, and $c$ knows $a$. In the graph shown above, there are two such triangles: (0, 1, 3) and (0, 3, 5).

Here is one way to count triangles, which uses linear algebra.

First, let $A \cdot B$ denote matrix multiplication. That is, $C = A \cdot B$ means $c_{i,j} = \sum_k a_{i,k} b_{k,j}$.

Next, let $A \odot B$ denote *elementwise* multiplication. That is, $E = A \odot B$ means $e_{i,j} = a_{i,j} b_{i,j}$.

Then, here is a two-step method to compute the number of triangles $t(A)$ in the graph:

$$C = (A \cdot A) \odot A$$

$$t(A) = \frac{1}{6} \sum_{i,j} c_{i,j}.$$

The first step computes a "count" matrix $C$. Each element, $c_{i,j}$, counts the number of triangles in which both $i$ and $j$ appear. For the example shown above, it turns out that $c_{0,1} = c_{1,0} = 1$ since there is only one triangle that uses the edge $(0, 1)$, whereas $c_{0,3} = c_{3,0} = 2$ because the edge $(0, 3)$ appears in two triangles.

The second step sums all the elements of $C$. However, because $c_{i,j} = c_{j,i}$ and, for any triangle $(i, j, k)$, the matrix $C$ includes a count for all three nodes, the matrix $C$ will overcount the number of unique triangles by a factor of six, hence the normalization of $\frac{1}{6}$. (One could instead just sum over the upper or lower triangle of $C$ since $C = C^T$, but more on that later!)

**Exercise 0** (3 points). Implement a function, **`count_triangles(A)`**, that implements the above formula. That is, given a symmetric Numpy array `A` representing the adjacency matrix of a graph, this function will return the number of triangles.

Your implementation should return a value of type **int**. For example, for the matrix in the sample,

```
assert count_triangles(A) == int(2)
```

In [1]:

Student's answer (Top)

```
import numpy as np

def count_triangles(A):
    assert (type(A) is np.ndarray) and (A.ndim == 2) and (A.shape[0] == A.shape[1])
    ### BEGIN SOLUTION
```

```
    return int(np.sum(np.multiply(A.dot(A), A)) / 6)
    ### END SOLUTION

A = np.array([[0, 1, 0, 1, 0, 1],
              [1, 0, 0, 1, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [1, 1, 1, 0, 1, 1],
              [0, 0, 0, 1, 0, 0],
              [1, 0, 0, 1, 0, 0]], dtype=int)
print(count_triangles(A))
```

```
2
```

In [2]:

Grade cell: `count_triangles_test` Score: 1.0 / 1.0 (Top)

```
# Test cell: `count_triangles_test`

ntri_A = count_triangles(A)
assert type(ntri_A) is int, "You should return a value of type `int`."
assert ntri_A == 2, "The graph only has 2 triangles, not {}".format(ntri_A)

print("\n(Passed part 1.)")
```

```
(Passed part 1.)
```

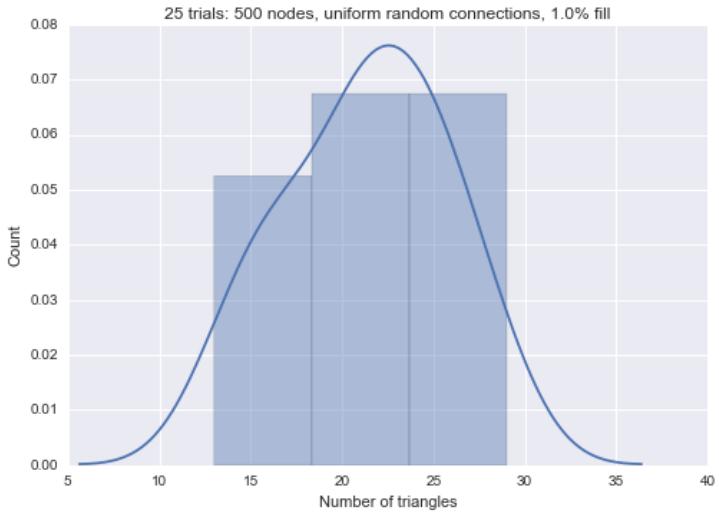
In [3]:

Grade cell: `count_triangles_test2` Score: 2.0 / 2.0 (Top)

```
# Test cell: `count_triangles_test2`
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

def check_count_triangles_large(n, den=1e-2):
    U_large = np.triu(np.random.rand(n, n) <= den).astype(int)
    np.fill_diagonal(U_large, 0)
    A_large = U_large + U_large.T
    return count_triangles(A_large)

n, den, k_max, mu, sd = 500, 1e-2, 25, 21, 5
nts = np.zeros(k_max, dtype=int)
for k in range(k_max):
    nts[k] = check_count_triangles_large(n, den)
    print(k, nts[k], np.sum(nts[:k+1])/(k+1))
sns.distplot(nts)
plt.xlabel("Number of triangles")
plt.ylabel("Count")
plt.title("{} trials: {} nodes, uniform random connections, {}% fill".format(k_max, n,
den*100))

assert (mu-sd) <= np.mean(nts) <= (mu+sd), "mean={}".format(np.mean(nts))
```

```
0 17 17.0

1 17 17.0

2 24 19.3333333333

3 13 17.75

4 25 19.2

5 25 20.1666666667
6 25 20.8571428571

7 14 20.0

8 23 20.3333333333

9 15 19.8

10 27 20.4545454545

11 20 20.4166666667
```

```
12 22 20.5384615385

13 17 20.2857142857

14 22 20.4

15 24 20.625

16 26 20.9411764706

17 20 20.8888888889

18 20 20.8421052632

19 21 20.85

20 16 20.619047619

21 29 21.0

22 28 21.3043478261

23 22 21.3333333333

24 21 21.32

/Users/richie/anaconda/lib/python3.5/site-packages/statsmodels/nonparametric/kdetools.py:
20: VisibleDeprecationWarning: using a non-integer number instead of an integer will resu
lt in an error in the future
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```




## Actor network

Let's apply the triangle counts to a real "social" network, namely, the graph of actors who have appeared in the same movie. The dataset in this problem uses data collected on a crawl of the Top 250 movies on the Internet Movie Database (https://github.com/napsternxg/IMDB-Graph/tree/master/Data/tutorial/tutorial) (circa 2012).

Let's start by loading this data, which is contained in a pair of JSON files.

In [4]:
```
import os
import json

def on_vocareum():
    return os.path.exists('.voc')

if on_vocareum():
    DATA_PATH = "../resource/lib/publicdata/imdb/"
else:
    DATA_PATH = "./imdb/"

assert os.path.exists(DATA_PATH), "Where are the data?"

def load_json(basefile, path=DATA_PATH):
    filename = basefile
```

```
    if path is not None: # Prepend non-empty paths
        filename = path + filename
    json_file = open(filename, encoding="utf8")
    json_str = json_file.read()
    json_data = json.loads(json_str)
    return json_data

movies_json = load_json("imdb.json")
casts_json = load_json("imdb_cast.json")
```

**About the data.** There are two parts to the data.

The first is `movies_json`, which is a JSON formatted collection of movie titles and IDs. It is a list and here are the first few entries:

In [5]:
```
print("=== First five entries of `movies_json` ===\n")
for k, m in enumerate(movies_json[:5]):
    print("[{}] {}".format(k, m))
print("...")
```

```
=== First five entries of `movies_json` ===

[0] {'link': ['/title/tt0111161/'], 'name': ['The Shawshank Redemption']}
[1] {'link': ['/title/tt0068646/'], 'name': ['The Godfather']}
[2] {'link': ['/title/tt0071562/'], 'name': ['The Godfather: Part II']}
[3] {'link': ['/title/tt0110912/'], 'name': ['Pulp Fiction']}
[4] {'link': ['/title/tt0060196/'], 'name': ['The Good, the Bad and the Ugly']}
...
```

The second part is `casts_json`, which is a JSON formatted collection of movies with information about who appeared in the movie. It is also a list and here is the very first element.

Observe that it is a dictionary with information about a single movie, including the movie's title, it's IMDB URL, and list of actors (cast members).

In [6]:
```
print("=== First entry of `casts_json` ===\n")
print(json.dumps(casts_json[0], indent=3))
```

```
=== First entry of `casts_json` ===

{
   "movie": {
      "url": "http://www.imdb.com/title/tt0050083/fullcredits",
      "name": [
         "12 Angry Men (1957) - Full cast and crew"
      ]
   },
   "rank": 1,
   "cast": [
      {
         "link": [
            "/name/nm0000842/"
         ],
         "name": [
            "Martin Balsam"
         ]
      },
      {
         "link": [
            "/name/nm0275835/"
         ],
         "name": [
            "John Fiedler"
         ]
      },
      {
         "link": [
            "/name/nm0002011/"
         ],
         "name": [
            "Lee J. Cobb"
         ]
      }
```

```
},
{
   "link": [
      "/name/nm0550855/"
   ],
   "name": [
      "E.G. Marshall"
   ]
},
{
   "link": [
      "/name/nm0001430/"
   ],
   "name": [
      "Jack Klugman"
   ]
},
{
   "link": [
      "/name/nm0083081/"
   ],
   "name": [
      "Edward Binns"
   ]
},
{
   "link": [
      "/name/nm0912001/"
   ],
   "name": [
      "Jack Warden"
   ]
},
{
   "link": [
      "/name/nm0000020/"
   ],
   "name": [
      "Henry Fonda"
   ]
},
{
   "link": [
      "/name/nm0842137/"
   ],
   "name": [
      "Joseph Sweeney"
   ]
},
{
   "link": [
      "/name/nm0003225/"
   ],
   "name": [
      "Ed Begley"
   ]
},
{
   "link": [
      "/name/nm0903667/"
   ],
   "name": [
      "George Voskovec"
   ]
},
{
   "link": [
      "/name/nm0916434/"
   ],
   "name": [
      "Robert Webber"
   ]
},
{
   "link": [
      "/name/nm0094036/"
```

```
                ],
                "name": [
                    "Rudy Bond"
                ]
            },
            {
                "link": [
                    "/name/nm0446450/"
                ],
                "name": [
                    "James Kelly"
                ]
            },
            {
                "link": [
                    "/name/nm0625180/"
                ],
                "name": [
                    "Billy Nelson"
                ]
            },
            {
                "link": [
                    "/name/nm0767847/"
                ],
                "name": [
                    "John Savoca"
                ]
            }
        ]
}
```


**Exercise 1** (2 points). Implement a function that, given the `casts_json` list, returns a dictionary that maps actor links to actor names.

In the example above, the first actor listed for "12 Angry Men" is `"Martin Balsam"`, whose link is `"/name/nm0000842/"`. Therefore, the dictionary that your function returns should include the key-value pair, `"/name/nm0000842/" : "Martin Balsam"`.

> *Slightly tricky bit.* You will need to pay careful attention to the structure of the output above to get this one right.

In [7]:

Student's answer (Top)

```
def gather_actors(casts):
    actors = dict() # Use to store (actor link) : (actor name) pairs
    for movie in casts:
        assert "cast" in movie
        ### BEGIN SOLUTION
        for actor in movie["cast"]:
            link = actor["link"][0]
            name = actor["name"][0]
            actors[link] = name
        ### END SOLUTION
    return actors

actors = gather_actors(casts_json)
print("Found {} unique actors.\n".format(len(actors)))

assert "/name/nm0000842/" in actors
print("'{}' -> '{}'".format("/name/nm0000842/", actors["/name/nm0000842/"]))
assert actors["/name/nm0000842/"] == "Martin Balsam"
```

```
Found 12975 unique actors.

'/name/nm0000842/' -> 'Martin Balsam'
```

In [8]:

Grade cell: `gather_actors_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `gather_actors_test`
```

```
assert ("/name/nm0872820/" in actors) and (actors["/name/nm0872820/"] == "Amedeo Trilli
")
assert ("/name/nm0279786/" in actors) and (actors["/name/nm0279786/"] == "Shug Fisher")
assert ("/name/nm0802831/" in actors) and (actors["/name/nm0802831/"] == "Tony Sirico")
assert ("/name/nm0924692/" in actors) and (actors["/name/nm0924692/"] == "Dean White")
assert ("/name/nm0248074/" in actors) and (actors["/name/nm0248074/"] == "Jake Eberle")
assert ("/name/nm1067542/" in actors) and (actors["/name/nm1067542/"] == "Grace Keller"
)
assert ("/name/nm0903694/" in actors) and (actors["/name/nm0903694/"] == "Carl Voss")
assert ("/name/nm1504897/" in actors) and (actors["/name/nm1504897/"] == "Radka Kucharo
va")
assert ("/name/nm0644905/" in actors) and (actors["/name/nm0644905/"] == "Tae-kyung Oh"
)
assert ("/name/nm0727037/" in actors) and (actors["/name/nm0727037/"] == "Gary Riley")
assert ("/name/nm2006011/" in actors) and (actors["/name/nm2006011/"] == "Glenn Stanton
")
assert ("/name/nm0193389/" in actors) and (actors["/name/nm0193389/"] == "John Curtis")
assert ("/name/nm0829189/" in actors) and (actors["/name/nm0829189/"] == "Avril Stewart
")
assert ("/name/nm1211469/" in actors) and (actors["/name/nm1211469/"] == "Karine Asure"
)
assert ("/name/nm0598388/" in actors) and (actors["/name/nm0598388/"] == "Jacques Monod
")
assert ("/name/nm1663820/" in actors) and (actors["/name/nm1663820/"] == "Michael Garne
t Stewart")
assert ("/name/nm0009388/" in actors) and (actors["/name/nm0009388/"] == "Khosrow Abris
hami")
assert ("/name/nm0020513/" in actors) and (actors["/name/nm0020513/"] == "Fletcher Alle
n")
assert ("/name/nm0615419/" in actors) and (actors["/name/nm0615419/"] == "John Murtagh"
)
assert ("/name/nm0120165/" in actors) and (actors["/name/nm0120165/"] == "Keith S. Bull
ock")
assert ("/name/nm0448560/" in actors) and (actors["/name/nm0448560/"] == "Colin Kenny")
assert ("/name/nm0882139/" in actors) and (actors["/name/nm0882139/"] == "David Ursin")
assert ("/name/nm1597244/" in actors) and (actors["/name/nm1597244/"] == "Carol Meirell
es")
assert ("/name/nm0316079/" in actors) and (actors["/name/nm0316079/"] == "Paul Giamatti
")
assert ("/name/nm3546231/" in actors) and (actors["/name/nm3546231/"] == "Leonard B. Jo
hn")

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 2** (2 points). Implement a function, **`count_appearances(casts)`**, that counts how many times each actor appeared in a movie. It should return a dictionary where the key is the actor's link and the value is the number of occurrences.

In [9]:

Student's answer (Top)

```
def count_appearances(casts):
    ### BEGIN SOLUTION
    appearances = {}
    for movie in casts:
        assert "cast" in movie
        for actor in movie["cast"]:
            link = actor["link"][0]
            if link in appearances:
                appearances[link] += 1
            else:
                appearances[link] = 1
    return appearances
    ### END SOLUTION

appearances = count_appearances(casts_json)
print("{} ({}) appeared in {} movies, according to your calculation.".format(actors['/n
ame/nm0000151/'],
                                                                             '/name/nm0
000151/',
                                                                             appearance
s['/name/nm0000151/']))
```

```
# Top 25 actors/actresses by number of appearances:
top25_appearances_links = sorted(appearances.keys(), key=appearances.__getitem__, rever
se=True)[:25]
top25_appearances = [(link, actors[link], appearances[link]) for link in top25_appearan
ces_links]
top25_appearances
```

Morgan Freeman (/name/nm0000151/) appeared in 7 movies, according to your calculation.

Out[9]:
```
[('/name/nm0283170/', 'Bess Flowers', 12),
 ('/name/nm0001652/', 'John Ratzenberger', 10),
 ('/name/nm0000071/', 'James Stewart', 9),
 ('/name/nm0000033/', 'Alfred Hitchcock', 9),
 ('/name/nm0000134/', 'Robert De Niro', 9),
 ('/name/nm0180679/', 'Gino Corrado', 8),
 ('/name/nm0528802/', 'Sherry Lynn', 8),
 ('/name/nm0569680/', 'Mickie McGowan', 7),
 ('/name/nm0000148/', 'Harrison Ford', 7),
 ('/name/nm0000151/', 'Morgan Freeman', 7),
 ('/name/nm0602100/', 'Bert Moorhouse', 6),
 ('/name/nm0003424/', 'Hank Mann', 6),
 ('/name/nm0000114/', 'Steve Buscemi', 6),
 ('/name/nm0915989/', 'Hugo Weaving', 6),
 ('/name/nm0000142/', 'Clint Eastwood', 6),
 ('/name/nm0000168/', 'Samuel L. Jackson', 6),
 ('/name/nm0000704/', 'Elijah Wood', 6),
 ('/name/nm0869863/', 'Arthur Tovey', 6),
 ('/name/nm0000122/', 'Charles Chaplin', 6),
 ('/name/nm0639444/', "William H. O'Brien", 6),
 ('/name/nm0000323/', 'Michael Caine', 6),
 ('/name/nm0785227/', 'Andy Serkis', 5),
 ('/name/nm0722636/', 'John Rhys-Davies', 5),
 ('/name/nm0828260/', 'Bert Stevens', 5),
 ('/name/nm0000027/', 'Alec Guinness', 5)]
```

In [10]:

Grade cell: `count_appearances_test` Score: 2.0 / 2.0 (Top)

```
# Test cell: `count_appearances_test`

assert appearances['/name/nm4723252/'] == 1 # Shirin Azimiyannezhad
assert appearances['/name/nm0574436/'] == 1 # Hilton McRae
assert appearances['/name/nm1753600/'] == 2 # Maciej Kowalewski
assert appearances['/name/nm1595614/'] == 1 # Diego Batista
assert appearances['/name/nm0201349/'] == 1 # Daphne Darling
assert appearances['/name/nm0642138/'] == 1 # Dennis O'Neill
assert appearances['/name/nm3325895/'] == 1 # Kyle Patrick Brennan
assert appearances['/name/nm0660998/'] == 1 # Charles Paraventi
assert appearances['/name/nm0703600/'] == 1 # Eddie Quillan
assert appearances['/name/nm0629697/'] == 1 # Rachel Nichols
assert appearances['/name/nm2715776/'] == 1 # Sharlene Grover
assert appearances['/name/nm0027323/'] == 1 # Richard Anderson
assert appearances['/name/nm0154021/'] == 2 # Geoffrey Chater
assert appearances['/name/nm0180987/'] == 1 # Lloyd Corrigan
assert appearances['/name/nm5523580/'] == 1 # Madhav Vaze
assert appearances['/name/nm0798620/'] == 1 # Ruth Silveira
assert appearances['/name/nm3193186/'] == 1 # Rubina Ali
assert appearances['/name/nm0361876/'] == 1 # Nigel Harbach
assert appearances['/name/nm0560983/'] == 1 # Michael Mauree
assert appearances['/name/nm0665886/'] == 2 # Lee Patrick
assert appearances['/name/nm0676349/'] == 1 # Brock Peters
assert appearances['/name/nm4587948/'] == 1 # Francesca Ortenzio
assert appearances['/name/nm0366873/'] == 1 # Neill Hartley
assert appearances['/name/nm0219666/'] == 1 # Reginald Denny
assert appearances['/name/nm2803526/'] == 1 # Don Kress

print("\n(Passed.)")
```

```
(Passed.)
```

**Exercise 3** (3 points). Implement a function that, for each actor, determines the number of triangles containing him or her. Store this result in

**Exercise 3** (5 points). Implement a function that, for each actor, determines the number of triangles containing him or her. Store this result in a dictionary named **`tri_counts`**, where **`tri_counts[link]`** stores the triangle count for the actor whose link is given by `link`.

> *Remark 0.* For this problem, the graph should be defined as follows. Each actor is a node; an edge exists between two actors if they appeared in the same movie.
>
> *Remark 1.* The choice of method to solve this exercise is open-ended. We have suggested the linear algebraic technique shown above; however, feel free to use any other computational approach that makes sense to you and can be implemented with reasonable efficiency -- see below.
>
> *Remark 2.* Whatever method you choose, it will be critical to exploit the sparsity of the actor network. That is, observe that there are nearly $n = 13,000$ actors in this dataset, so any method that scales much worse than $\mathcal{O}(n^2)$ will not pass the autograder's timing test.

In [11]:

Student's answer (Top)

```
### BEGIN SOLUTION

def assign_actor_ids(actors):
    id2actor = {}
    actor2id = {}
    for i, link in enumerate(actors.keys()):
        id2actor[i] = link
        actor2id[link] = i
    return id2actor, actor2id

def build_cast_matrix(casts, actor2id):
    from itertools import combinations
    from scipy.sparse import coo_matrix

    # Build coordinate representation
    I = []
    J = []
    for movie in casts:
        for ai, aj in combinations(movie["cast"], 2):
            i = actor2id[ai["link"][0]]
            j = actor2id[aj["link"][0]]
            I.append(i)
            J.append(j)
            I.append(j)
            J.append(i)

    # Convert into a sparse matrix!
    n = max(I) + 1
    V = np.ones(len(I), dtype=int)
    A = coo_matrix((V, (I, J)), shape=(n, n))
    A.sum_duplicates()
    A = (A > 0).astype(int)
    return A

def count_actor_triangles(A):
    return A.dot(A) * A

def sum_actor_triangles(C, id2actor):
    n = len(id2actor)
    tri_counts_raw = C.sum(axis=1)
    tri_counts = {}
    for i, c in enumerate(tri_counts_raw):
        tri_counts[id2actor[i]] = c.item()
    return tri_counts

# === Solution using the above as building blocks ===
actors = gather_actors(casts_json)
id2actor, actor2id = assign_actor_ids(actors)
A = build_cast_matrix(casts_json, actor2id)
C = count_actor_triangles(A)
tri_counts = sum_actor_triangles(C, id2actor)

print("Total number of triangles: ~ {:.1f} billion".format((C.sum() / 6 * 1e-9)))
### END SOLUTION
```

```
Total number of triangles: ~ 5.4 billion
```

In [12]:

Grade cell: `tri_counts_test0` Score: 1.0 / 1.0 (Top)

```
# Test cell: `tri_counts_test0` -- 1 point for computing something

# From https://docs.python.org/3/howto/sorting.html
assert type(tri_counts) is dict, "Per the instructions, `tri_counts` should be a dicton
ary."
top10_actor_links = sorted(tri_counts.keys(), key=tri_counts.__getitem__, reverse=True)
[:10]
top10_actor_counts = [(link, actors[link], tri_counts[link]) for link in top10_actor_li
nks]
top10_actor_names = [n for _, n, _ in top10_actor_counts]
print("Actors in the most triangles:\n{}".format(top10_actor_counts))

print("\n(Passed part 1 of 3.)")
```

```
Actors in the most triangles:
[('/name/nm0000198/', 'Gary Oldman', 38289360), ('/name/nm0180679/', 'Gino Corrado', 3820
0470), ('/name/nm0000151/', 'Morgan Freeman', 37309136), ('/name/nm0000323/', 'Michael Ca
ine', 36764575), ('/name/nm0000288/', 'Christian Bale', 36468321), ('/name/nm0614165/', '
Cillian Murphy', 36404837), ('/name/nm0175410/', "Frank O'Connor", 36370907), ('/name/nm2
720736/', 'Ernest Pierce', 35835774), ('/name/nm0602100/', 'Bert Moorhouse', 34232302), (
'/name/nm0227117/', 'John Dilson', 33411930)]

(Passed part 1 of 3.)
```

In [13]:

Grade cell: `tri_counts_test1` Score: 1.0 / 1.0 (Top)

```
# Test cell: `tri_counts_test1` -- 2 points for getting at least half the top actors

our_top10 = {'Bert Moorhouse', 'Christian Bale', 'Cillian Murphy', \
             'Ernest Pierce', "Frank O'Connor", 'Gary Oldman', \
             'Gino Corrado', 'Michael Caine', 'Milton Kibbee', \
             'Morgan Freeman'}
your_top10 = set(top10_actor_names)
assert len(our_top10 & your_top10) >= 5, "Got less than half the top 10 right."

print("\n(Passed part 2 of 3.)")
```

```
(Passed part 2 of 3.)
```

In [14]:

Grade cell: `tri_counts_test2` Score: 1.0 / 1.0 (Top)

```
# Test cell: `tri_counts_test2` -- 1 point for getting the rank-ordered top 10 list exa
ctly

our_top10_list_weighted = ['Gino Corrado', "Frank O'Connor", 'Bert Moorhouse', 'Gary Ol
dman',
                           'Michael Caine', 'Morgan Freeman', 'Cillian Murphy', 'Christ
ian Bale',
                           'Milton Kibbee', 'Ernest Pierce']
our_top10_list_unweighted = ['Gary Oldman', 'Gino Corrado', 'Morgan Freeman', 'Michael
Caine',
                             'Christian Bale', 'Cillian Murphy', "Frank O'Connor", 'Ern
est Pierce',
                             'Bert Moorhouse', 'John Dilson']

# Meta-comment: The top actors by number of appearances includes men and women,
# but the actors that appear in the most triangles seems to be dominated by men!

our_top_list_merged = set(our_top10_list_weighted) | set(our_top10_list_unweighted)

assert len(set(our_top_list_merged) & set(top10_actor_names)) >= 9, "Too many mismatche
s in your top 10 list"
print("\n(Passed part 3 of 3.)")
```

```
(Passed part 3 of 3.)
```

(Passed part 5 of 5.)

**Fin!** That's the end of this problem, which is the last problem of the final exam. Don't forget to submit your work!