In [4]:
```
import time
import itertools
import os
import numpy as np
import pandas as pd

import conceptnet_lite
from conceptnet_lite import Label, edges_between, edges_for
```

In [56]:
```
rat = 'RAT.csv'
frat = 'fRAT.csv'

### set the csv file to examine
csv = rat
```

In [57]:
```
if csv == rat:
    # in csv format
    df = pd.read_csv(os.path.join('input', csv))
else:
    df = pd.read_csv(os.path.join('input', csv), sep=';')

conceptnet_lite.connect(r"C:\Users\rejna\Work_only_here\Miscellaneous\rakshitha\conceptnet_database")

print('***************    CSV file set to \033[1m {} \033[0;0m    ***************'.format(csv[:-4]))
df.head()
```

```
***************    CSV file set to  RAT      ***************
```

Out[57]:

	w1	w2	w3	wans
0	Cottage	Swiss	Cake	Cheese
1	Cream	Skate	Water	Ice
2	Loser	Throat	Spot	Sore
3	Show	Life	Row	Boat
4	Night	Wrist	Stop	Watch

In [58]:
```
# new_df = pd.DataFrame(df.RAT.str.split(' ').tolist(),
#                       columns = ['w1','w2', 'w3', 'w4'])

# new_df.w4 = df.Solutions
# new_df.columns = ['w1', 'w2', 'w3', 'wans']
# new_df.to_csv('rat.csv', index=False)
```

In [59]:
```
def save_csv(content, random=''):
    output = pd.DataFrame(content)
    file_output = csv.strip('.csv') + random + '_check_' + check_for.strip(',') + '_solution_' + str(t)
 + '.xlsx'
    if not os.path.exists('output'):
        os.mkdir('output')

    output.to_excel(os.path.join('output', file_output), index=False)
```

FRAT & RAT

Local search implementation

Search for related nodes intersection (no compound words)

In [60]:
```
# Hyperparameters
check_for = '2,3' # separate digits by comma, even if only 1
t = True # whether or not the solution should be contained
```

In [61]:
```
def get_nodes_frat(node):
    """ Given a word, get all the words related to it

        Returns:
            dictionary of form:
                {"pick_someone's_brain": ['related_to'], 'blindly': ['related_to'], ...  'cross_purpos
e': ['related_to']}
    """
    try:
        # TODO: is there a problem with the relationship's direction here?
        current = [(e.start.text, e.relation.name)
                   for e in edges_for(Label.get(text=node, language='en').concepts, same_language=True
)
                       if e.start.text not in [node]]
        [current.append((e.end.text, e.relation.name))
             for e in edges_for(Label.get(text=node, language='en').concepts, same_language=True)
                 if e.end.text not in [node]]

        result = {}
        for tup in list(set(current)):
            if tup[0] not in result:
                result[tup[0]] = list()
                result[tup[0]].append(tup[1])
            else:
                result[tup[0]].append(tup[1])
        return result
    except Exception as error:
        print('No label for the node "{}"... Are you sure the spelling is correct?'.format(node))
        return {}

def get_nodes_rat(word):
    """ Given a word, get all the compound words related to it as well as their relation name
        Compound words are basically being identified by the underscore (_)
    """
    result = []
    relation = []
    for e in edges_for(Label.get(text=word).concepts, same_language=True):
        if (e.start.text.find('_') != -1) & (e.start.text.find(word) != -1):
            result.append(e.start.text.replace(word, '').strip('_'))
            relation.append(e.relation.name)
        if (e.end.text.find('_') != -1) & (e.end.text.find(word) != -1):
            result.append(e.end.text.replace(word, '').strip('_'))
            relation.append(e.relation.name)

    joint_result = []
    for i in range(len(result)):
        if result[i].find('_') != -1:
            words = result[i].split('_')
            for word in words:
                if word != '': joint_result.append((word, relation[i]))
        else:
            joint_result.append((result[i], relation[i]))

#     return joint_result

    final_result = {}
    for tup in list(set(joint_result)):
        if tup[0] not in final_result:
            final_result[tup[0]] = list()
            final_result[tup[0]].append(tup[1])
        else:
            final_result[tup[0]].append(tup[1])
    return final_result
    # words can still be compounded, so we split them and merge the lists
#     return list(itertools.chain(*[filter(len, word.split('_')) for word in result])), relation
```

In [62]:
```
def checker(relation_dict, check_for, cue):
    results = [set(relation_dict[key].keys()) for key in relation_dict.keys()]

    if '3' in check_for:
        yield results[0] & results[1] & results[2], relation_dict, [cue[0], cue[1], cue[2]]
    if '2' in check_for:
        yield results[0] & results[1], relation_dict, [cue[0], cue[1]]
        yield results[0] & results[2], relation_dict, [cue[0], cue[2]]
        yield results[1] & results[2], relation_dict, [cue[1], cue[2]]

def get_output(result, cues, relation_dict, solution, has_solution):
    solutions = [res for res in result] if result else []
    relations = list()

    # build a relationship message for each (1) node, (2) relation (3) solution
    # For example:
    #
    # cues: antlers, doe, fawn
    # relation: related_to
    # solution: deer
    # relationship message: antler is related_to deer, doe is related_to to deer, fawn is related_to to
 deer
    for cue in cues:
        for sol in solutions:
            rel = ', '.join(relation_dict[cue][sol.strip()]) # get the relationships for each cue and s
olution
            relations.append(cue + ' is "'+ rel + '" to ' + sol)

    return {'FrAt': ', '.join(cues),
            'ground solution': solution,
            'solutions': ', '.join(solutions),
            'has_solution': has_solution,
            'relation': ' | '.join(relations)
           }
```

In [63]:
```
concat = df.w1 + ' ' + df.w2 + ' ' + df.w3
concat = concat[:-1] # remove last nan element
cues = [list(map(lambda x: x.lower(), filter(len, line.split(' ')))) for line in concat]
get_nodes = get_nodes_rat if csv == rat else get_nodes_frat
start_time = time.time()
output = []
index = 0
total = 0
tp = 0

for cue in cues:
    results = {}
    solution = df.iloc[index].wans
    index +=1
    print('Finished {}. Timestamp: {} min'.format(cue, round((time.time()-start_time)/60, 2)))

    for c in cue:
        results[c] = get_nodes(c)

    # so the format of the results dictionary at this point would be
    #
    # {'question': {"pick_someone's_brain": ['related_to'], 'blindly': ['related_to'], ... 'cross_purpo
se': ['related_to']},
    # 'reply': {'repone': ['related_to'], ... 'sentences': ['related_to']},
    # 'solutionst': {'solutionsn': ['derived_from', 'related_to'],... 'exhibit': ['related_to']}}

    for result, relation_dict, cue in checker(results, check_for, cue):
        total += 1
        has_solution = any(solution.lower().strip() in res for res in result)

        if has_solution: tp+=1

        output.append(get_output(result, cue, relation_dict, solution, has_solution))


# save
output.append({'Accuracy': str(round(100*tp/total, 2)) + '%'})
save_csv(output, '_new_2')
```

```
Finished ['cottage', 'swiss', 'cake']. Timestamp: 0.0 min
Finished ['cream', 'skate', 'water']. Timestamp: 0.04 min
Finished ['loser', 'throat', 'spot']. Timestamp: 0.12 min
Finished ['show', 'life', 'row']. Timestamp: 0.15 min
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: