```
import csv
from itertools import combinations as cb
import os

trans_list = []
freq_items = []
num_of_cbs = 1
transaction_rows = 0

ip_trs = []
for file in os.listdir('./'):
    if file.startswith("tr"):
        ip_trs.append(file)

print('Following is a list of transactions files. Choose the one you want
 to implement Apriori Algorithm on: ')
for i in ip_trs:
    print(i)

file_name = str(input('Enter the name of the file you want to process: ')
)

print(" ")

fob = open(file_name, 'r', encoding='ascii')
reader = csv.reader(fob)

print('**************************** File: ', file_name,
' ****************************')
print(" ")

for row in reader:
    trans_list.append(row)
    transaction_rows += 1

print('***************** TRANSACTIONS IN ', file_name, ' ARE AS FOLLOWS
 ****************')
print(" ")

for row in trans_list:
    print(row)
print(" ")
print('************************ TRANSACTIONS ARE ABOVE
 *****************************')
print(" ")

user_def_supp = int(input("Please Enter Support Value in % : "))
# user_def_supp = 20
user_def_confidence = int(input("Please Enter Confidence Value in % : "))
# user_def_confidence = 50
total_trans = transaction_rows  # Total num of transactions

print('********************** USER DEFINED ATTRIBUTES BELOW
  ***********************')
print(" ")
print('\t', '\t', 'SUPPORT: ', user_def_supp, '%', '\t\t\t\t\t',  '
CONFIDENCE: ', user_def_confidence, '%')
print(" ")
print('********************** USER DEFINED ATTRIBUTES ABOVE
```

```
 **********')

# ================================================================================
==============


def calc_freq_items(items_1):
    global freq_items
    global user_def_supp
    global num_of_cbs

    # All possible num_of_cbs item combinations from items_1
    items_cb = []
    for item in cb(items_1, num_of_cbs):
        items_cb.append(list(item))

    # Converting list of lists of lists into list of lists
    items_cb_mod_s = []
    for i in items_cb:
        if isinstance(i[0], list):
            items_cb_sub = [k for j in i for k in j]
        else:
            items_cb_mod_s = items_cb[:]
            break
        if len(items_cb_sub) != 0:
            items_cb_mod_s.append(items_cb_sub)

    items_cb_mod_d = []
    for i in items_cb_mod_s:
        items_cb_mod_d.append(list(set(i)))

    # removes duplicate combination lists from items_cb_mod_d
    items_cb_mod = [list(x) for x in set(tuple(x) for x in items_cb_mod_d
)]

    # print
('-------------------------------------------------------------')
    # print('items_cb_mod:', items_cb_mod)

    # count_items contains all the combinations the number of times they
appear in the transactions
    count_items = []
    for i in trans_list:
        for k in items_cb_mod:
            if set(k).issubset(set(i)):
                count_items.append(k)

    # num_items has its key equal to the index in items_cb_mod and val
    # equal to the count in transactions
    num_items = []
    for i in items_cb_mod:
        count = 0
        for j in count_items:
            if set(j) == set(i):
                count += 1
        num_items.append(count)

    # print
('-------------------------------------------------------------')
```

```
    # print('num_items: ', num_items)

    # Create a duplicate of the combinations list
    items_cb_mod_list = items_cb_mod[:]

    # Support Check
    del_items = []
    for i in items_cb_mod:
        if round((num_items[items_cb_mod.index(i)] / total_trans) * 100)
< user_def_supp:
            del_items.append(i)

    # print
('------------------------------------------------------------')
    # print('Delete these: ', del_items)

    # Remove elements with support less than user_def_supp from
items_cb_mod_list and num_items
    for j in del_items:
        del num_items[items_cb_mod_list.index(j)]
        items_cb_mod_list.remove(j)

    # print
('------------------------------------------------------------')
    # print('Final List: ', items_cb_mod_list)

    # Append freq sets to freq_items list
    for i in items_cb_mod_list:
        freq_items.append(i)

    if len(items_cb_mod_list) != 0:
        if num_of_cbs < 2:
            num_of_cbs += 1
        calc_freq_items(items_cb_mod_list)


# ================================================================================
===============
# The main function to generate frequent items
items_list = []  # List of all distinct items involved in transactions
for i in trans_list:
    for j in i:
        if j not in items_list:
            items_list.append(j)
calc_freq_items(items_list)

print(" ")
print('************************** FREQUENT ITEMS BELOW
 ****************************')
print(" ")
print(freq_items)
print(" ")
print('************************** FREQUENT ITEMS ABOVE
 ****************************')

# ================================================================================
===============
# ================================================================================
===============
```

```
# ============================================================================
===============
# ============================================================================
===============
# ============================================================================
===============

# Part 2 - ASSOCIATION RULES
************************************************************

# freq_items_mod has all freq item sets from which Association Rules can
be generated
freq_items_mod = []
for i in freq_items:
    if len(i) > 1:
        freq_items_mod.append(i)


# Takes LHS and RHS of Association rule and calculates confidence and
support of the rule
def calc_confidence(ping):
    lef = ping[0]
    lef_cp = lef[:]
    r = ping[1]
    lef_cp.extend(r)
    global total_trans
    sup_lef_cp = calc_support(lef_cp)
    sup_l = calc_support(lef)
    return [sup_lef_cp, round((sup_lef_cp/sup_l)*100, 1)]


# Takes a item set as argument and calculates its support
def calc_support(ip_list):
    count_lr = 0
    for i in trans_list:
        temp_list_lr = []
        for j in cb(i, len(ip_list)):
            temp_list_lr.append(list(j))
        for k in temp_list_lr:
            if set(ip_list) == set(k):
                count_lr += 1
            else:
                continue
    return count_lr


# generate association rules and return a list of lists of Association
rules
# for eg: [a,b,c] => [[[a],[b,c]], [[b],[a,c]], [[c],[a,d]]].......
def gen_assoc_rules(li):
    rules = []
    combs = []

    for p in range(1, len(li)):
        for c in cb(li, p):
            combs.append(list(c))

    for i in combs:
        left = i
```

```
        right = list(set(li)-set(i))
        rules.append([left, right])

    return rules


print(" ")
print('************************ ASSOCIATION RULES BELOW
 ***************************')
print(" ")

# Iterate over freq_item_mod to find association rules and corresponding
support and confidence values
a_rules = []
for f in freq_items_mod:
    assoc_rules = gen_assoc_rules(f)
    for a in assoc_rules:
        ret_get_conf = calc_confidence(a)
        sup = round((ret_get_conf[0]/total_trans)*100, 1)
        conf = ret_get_conf[1]
        if (sup >= user_def_supp) & (conf >= user_def_confidence):
            print(a[0], ' -> ', a[1], ' Support: ', sup, ' Confidence: ',
 conf)
            a_rules.append(a)

if len(a_rules) == 0:
    print('No Rules that satisfy the conditions')

print(" ")
print('************************ ASSOCIATION RULES ABOVE
 ***************************')
```