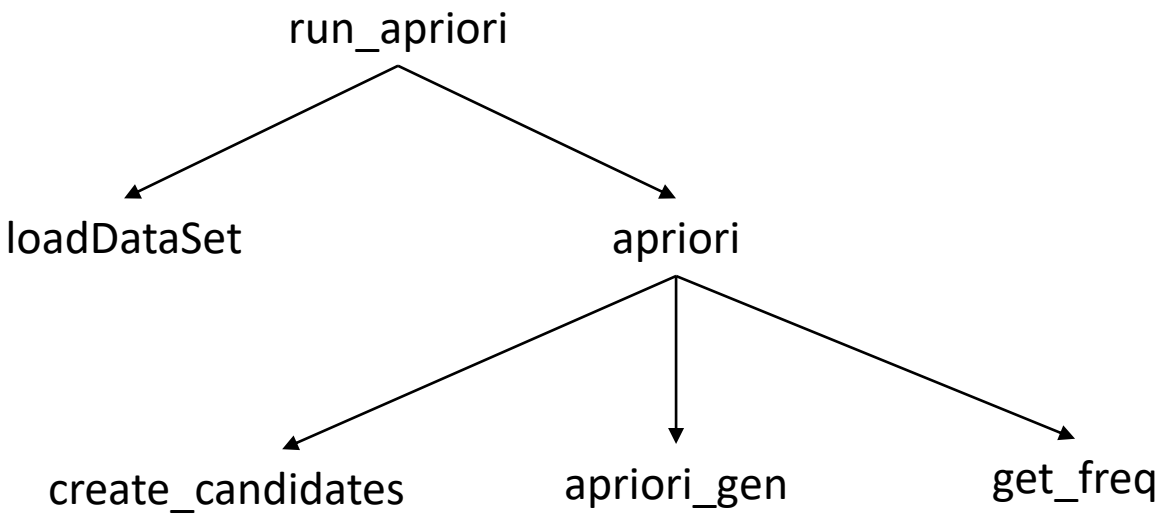
Apriori Templete

- Data
  - Market_data(shown in the picture)
  - Gene_data

```
Corn,Ice-cream,Key-chain,Mango,Umbrella,Yo-yoCRLF
Doll,Eggs,Key-chain,Mango,Onion,UmbrellaCRLF
Eggs,Ice-cream,Key-chain,Mango,Onion,Yo-yoCRLF
Corn,Eggs,Key-chain,Onion,Yo-yoCRLF
Apple,Doll,Mango,Nintendo,OnionCRLF
```

- Function ***loadDataSet*** is already implemented in the template

Apriori Templete



Function call relationship

Main function

```
if __name__ == '__main__':
    if len(sys.argv)==3:
        F, support = run_apriori(sys.argv[1], float(sys.argv[2]))
    elif len(sys.argv)==4:
        F, support = run_apriori(sys.argv[1], float(sys.argv[2]), bool_transfer(sys.argv[3]))
    else:
        raise ValueError('Usage: python apriori_templete.py <data_path> <min_support> <is_verbose>')
    print(F)
    print(support)

    '''
    Example:

    python apriori_templete.py market_data_transaction.txt 0.5

    python apriori_templete.py market_data_transaction.txt 0.5 True

    '''
```

How to run the file

Run_apriori

```
def run_apriori(data_path, min_support, verbose=False):
    dataset = loadDataSet(data_path)
    F, support = apriori(dataset, min_support=min_support, verbose=verbose)
    return F, support
```

Load dataset as list

Run apriori

Apriori

```
def apriori(dataset, min_support=0.5, verbose=False):
    C1 = create_candidates(dataset)
    D = list(map(set, dataset))
    F1, support_data = get_freq(D, C1, min_support, verbose=False) # get frequent 1-itemsets
    F = [F1] # list of frequent itemsets; initialized to frequent 1-itemsets
    k = 2 # the itemset cardinality
    while (len(F[k - 2]) > 0):
        Ck = apriori_gen(F[k-2], k) # generate candidate itemsets
        Fk, supK  = get_freq(D, Ck, min_support) # get frequent itemsets
        support_data.update(supK) # update the support counts to reflect pruning
        F.append(Fk)  # add the frequent k-itemsets to the list of frequent itemsets
        k += 1

    if verbose:
        # Print a list of all the frequent itemsets.
        for kset in F:
            for item in kset:
                print("                    " + "{                    " + "".join(str(i) + ", " for i in iter(item)).rstrip(', ')

    return F, support_data
```

the set of single item

the set of single item whose support is not less than minSupport

Detail

- ***update*** method
- You can use a.update(b) to add keys and values in dict b into dict a.

```
>>> dict = {'Name': 'Zara', 'Age': 7}
>>> dict2 = {'Sex': 'female' }
>>> dict.update(dict2)
>>> dict
{'Name': 'Zara', 'Age': 7, 'Sex': 'female'}
```

apriori_gen

- The apriori_gen function performs two operations:
  - Generate length k candidate itemsets from length k-1 frequent itemsets
  - Prune candidate itemsets containing subsets of length k-1 that are infrequent
- Input:
  - freq_sets: The list of frequent (k-1)-itemsets.
  - k: The cardinality of the current itemsets being evaluated.
- Output:
  - candidate_list : The list of candidate itemsets.

```
def apriori_gen(freq_sets, k):
```

apriori_gen

- It can be implemented following the pseudo-code.

//generate and prune candidate set C_k
C_k is a list of itemsets in which each itemset is formed by merging two itemsets in L_{k-1} if their first *k-2* items are identical
Remove an itemset from C_k if any (*k-1*)-subset of this candidate itemset is not in the frequent itemset list L_{k-1}

apriori_gen

- You may use ***combinations*** method to enumerate all the subsets.

```
>>> from itertools import combinations
>>> a = combinations('abcd',2)
>>> for i in a:
...     print(i)
...
('a', 'b')
('a', 'c')
('a', 'd')
('b', 'c')
('b', 'd')
('c', 'd')
```

- You must import itertools first!

get_freq

- Input
  - dataset: a list of transactions from which to generate candidate itemsets
  - candidates: the list of candidate itemsets
  - min_support
- Output
  - freq_list: the list of frequent itemsets
  - support_data: the support data for all candidate itemsets

```
def get_freq(dataset, candidates, min_support, verbose=False):
```

get_freq

- It can be implemented following the pseudo-code.

```
//Count the support of each candidate itemset
for each transaction t in database do{
        for each candidate c in C_k
            // increment the support count of all candidate itemsets that are
        contained in transaction t
          if  c is a subset of t then count[c] ← count[c] + 1
}
for each candidate c in C_k
            // Judge if a candidate itemset is frequent or not
          if  the support of c is not less than minSupport
                then include c in  L_k
```

get_freq

- You may use ***issubset*** method during support counting in get_freq.

```
>>> x = {"a", "b", "c"}
>>> y = {"f", "e", "d", "c", "b", "a"}
>>> z={"a","z"}
>>> x.issubset(y)
True
>>> z.issubset(y)
False
```