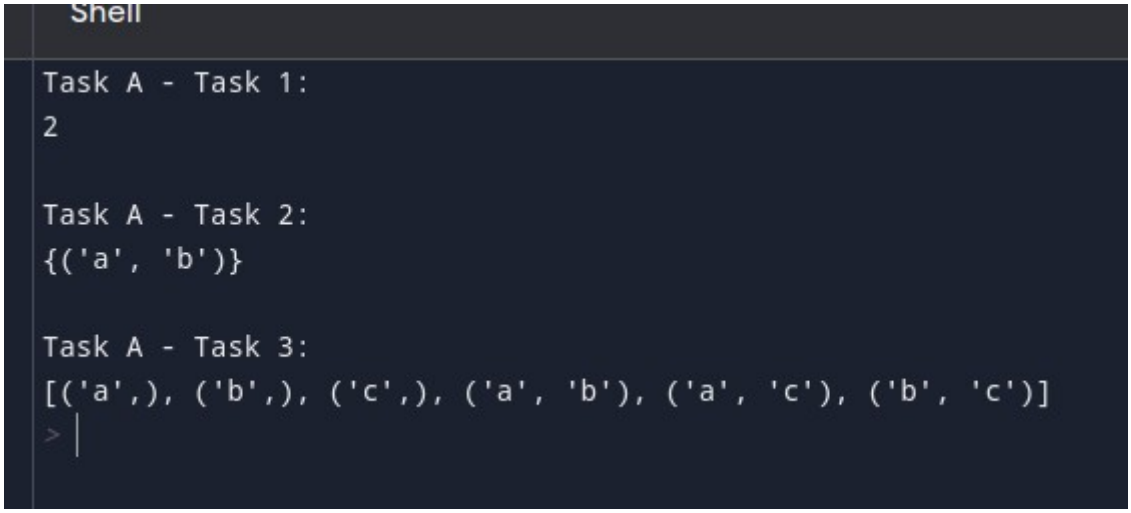
DWDM ASSIGNMENT – 5
21BCE7371
RADHA KRISHNA GARG

ANS -1

```
from itertools import combinations

# Task A

# Task 1
def count_sets(key, data):
    count = 0
    for transaction in data:
        if set(key).issubset(transaction):
            count += 1
    return count

# Task 2
def self_join(items, k):
    return set(combinations(sorted(items), k + 1))

# Task 3
def generate_subsets(items):
    subsets = []
    for i in range(1, len(items)):
        subsets.extend(combinations(items, i))
    return subsets
```

```
from itertools import combinations

# Task A

# Task 1
def count_sets(key, data):
    count = 0
    for transaction in data:
        if set(key).issubset(transaction):
            count += 1
    return count

# Task 2
def self_join(items, k):
    return set(combinations(sorted(items), k + 1))

# Task 3
def generate_subsets(items):
    subsets = []
    for i in range(1, len(items)):
        subsets.extend(combinations(items, i))
    return subsets

# Test Task A
print("Task A - Task 1:")
print(count_sets(('a', 'b', 'c'), [['a', 'b', 'c', 'd'], ['b', 'c', 'd'], ['b', 'c', 'd', 'e'], ['a',
    'b', 'c', 'd', 'e']]))

print("\nTask A - Task 2:")
print(self_join(('a', 'b'), 1))

print("\nTask A - Task 3:")
print(generate_subsets(['a', 'b', 'c']))
```

OUTPUT



ANS-2

```
# Task B

def apriori(transactions, min_support_count, min_confidence):
    # Step 1: Generate frequent item sets of size 1
    all_items = set()
    for transaction in transactions:
        all_items.update(transaction)

    frequent_item_sets = []
    for item in all_items:
        count = count_sets((item,), transactions)
        if count >= min_support_count:
            frequent_item_sets.append(((item,), count))

    # Step 2: Generate frequent item sets of size > 1 using the Apriori property
    k = 2
    while frequent_item_sets:
        next_candidates = set()
        for item_set, _ in frequent_item_sets:
            for item in all_items:
                if item not in item_set:
                    new_set = tuple(sorted(list(item_set) + [item]))
                    if new_set not in next_candidates and self_join(item_set, k -
1).issubset(frequent_item_sets):
                        next_candidates.add(new_set)
        frequent_item_sets = []
        for candidate in next_candidates:
            count = count_sets(candidate, transactions)
            if count >= min_support_count:
                frequent_item_sets.append((candidate, count))
        k += 1
```

```
    # Step 3: Generate association rules
    association_rules = []
    for item_set, support_count in frequent_item_sets:
        for i in range(1, len(item_set)):
            for antecedent in combinations(item_set, i):
                antecedent = tuple(sorted(antecedent))
                consequent = tuple(sorted(set(item_set) - set(antecedent)))
                confidence = support_count / count_sets(antecedent, transactions)
                if confidence >= min_confidence:
                    association_rules.append(((antecedent, consequent), support_count, confidence))

    return association_rules
```

OUTPUT

```
# Task B

def apriori(transactions, min_support_count, min_confidence):
    # Step 1: Generate frequent item sets of size 1
    all_items = set()
    for transaction in transactions:
        all_items.update(transaction)

    frequent_item_sets = []
    for item in all_items:
        count = count_sets((item,), transactions)
        if count >= min_support_count:
            frequent_item_sets.append(((item,), count))

    # Step 2: Generate frequent item sets of size > 1 using the Apriori property
    k = 2
    while frequent_item_sets:
        next_candidates = set()
        for item_set, _ in frequent_item_sets:
            for item in all_items:
                if item not in item_set:
                    new_set = tuple(sorted(list(item_set) + [item]))
                    if new_set not in next_candidates and self_join(item_set, k - 1).issubset
                        (frequent_item_sets):
                        next_candidates.add(new_set)
        frequent_item_sets = []
        for candidate in next_candidates:
            count = count_sets(candidate, transactions)
            if count >= min_support_count:
                frequent_item_sets.append((candidate, count))
        k += 1

    # Step 3: Generate association rules
```

```
    k += 1

    # Step 3: Generate association rules
    association_rules = []
    for item_set, support_count in frequent_item_sets:
        for i in range(1, len(item_set)):
            for antecedent in combinations(item_set, i):
                antecedent = tuple(sorted(antecedent))
                consequent = tuple(sorted(set(item_set) - set(antecedent)))
                confidence = support_count / count_sets(antecedent, transactions)
                if confidence >= min_confidence:
                    association_rules.append(((antecedent, consequent), support_count, confidence))

    return association_rules

    # Test Task B
transactions = [
    ("a", "b", "c"),
    ("a", "b"),
    ("a", "b", "d"),
    ("b", "e"),
    ("b", "c", "e"),
    ("a", "d", "e"),
    ("a", "c"),
    ("a", "b", "d"),
    ("c", "e"),
    ("a", "b", "d", "e"),
    ("a", 'b', 'e', 'c')
]
min_support_count = 3
min_confidence = 0.8

print("\nTask B - Apriori Algorithm:")
print(apriori(transactions, min_support_count, min_confidence))
```