Create a knowledge base consisting of first order logic statements and prove the given query using Resolution

```
# resolution_prover.py
from copy import deepcopy


# --------------------------------
# Resolution helper: pair resolve
# --------------------------------
def resolve_pair(ci, cj):
    """
    Try to resolve two clauses (lists of literals).
    Returns a list of resolvent clauses.
    """
    resolvents = []
    for lit_i in ci:
        for lit_j in cj:
            # Case 1: ¬P vs P
            if lit_i.startswith('¬') and lit_i[1:] == lit_j:
                new_clause = list(set(ci + cj))
                new_clause.remove(lit_i)
                new_clause.remove(lit_j)
                resolvents.append(new_clause)

            # Case 2: P vs ¬P
            elif lit_j.startswith('¬') and lit_j[1:] == lit_i:
                new_clause = list(set(ci + cj))
                new_clause.remove(lit_j)
                new_clause.remove(lit_i)
                resolvents.append(new_clause)
    return resolvents
```

```
# ------------------------------
# Resolution algorithm
# ------------------------------
def resolution(KB, query):
    """
    Apply the resolution method to KB ∪ {¬query}.
    KB is a list of clauses (each clause = list of literals).
    """
    clauses = deepcopy(KB)
    clauses.append(['¬' + query])  # Add negated query

    print("=== INITIAL CLAUSES ===")
    for c in clauses:
        print(c)

    new = set()

    while True:
        n = len(clauses)
        pairs = [(clauses[i], clauses[j]) for i in range(n) for j in range(i + 1, n)]

        for (ci, cj) in pairs:
            resolvents = resolve_pair(ci, cj)
            for res in resolvents:
                if not res:
                    print(f"\nResolved {ci} and {cj} -> []")
                    print("□ Empty clause derived ⇒ Query PROVED!")
                    return True
                new.add(tuple(sorted(res)))

        new_clauses = [list(x) for x in new if list(x) not in clauses]
```

```
        if not new_clauses:
            print("\nNo new clauses ⇒ Query cannot be proved.")
            return False

        for c in new_clauses:
            clauses.append(c)
            print("Added new clause:", c)



# ------------------------------
# Knowledge Base (Grounded)
# ------------------------------
KB = [
    ['¬Food(Apple)', 'Likes(John,Apple)'],        # John likes Apple if Apple is Food
    ['¬Food(Vegetable)', 'Likes(John,Vegetable)'],# John likes Vegetable if it's Food
    ['¬Food(Peanut)', 'Likes(John,Peanut)'],      # John likes Peanut if it's Food
    ['Food(Apple)'],                              # Apple is Food
    ['Food(Vegetable)'],                          # Vegetable is Food
    ['Alive(Anil)'],                              # Anil is Alive
    ['¬Alive(Anil)', 'NotKilled(Anil)'],          # Alive → NotKilled
    ['¬NotKilled(Anil)', 'Alive(Anil)'],          # NotKilled → Alive
    ['Eats(Anil,Peanut)'],                        # Anil eats Peanut
    ['¬Eats(Anil,Peanut)', '¬NotKilled(Anil)', 'Food(Peanut)']  # Eats & NotKilled → Food
]

query = 'Likes(John,Peanut)'



# ------------------------------
# Run Resolution
# ------------------------------
if __name__ == "__main__":
```

```
print(f"Proving query: {query}\n")
result = resolution(KB, query)


print("\n=== RESULT ===")
if result:
    print("□ The query is PROVED using resolution.")
else:
    print("□ The query CANNOT be proved from the KB.")
```

OUTPUT: