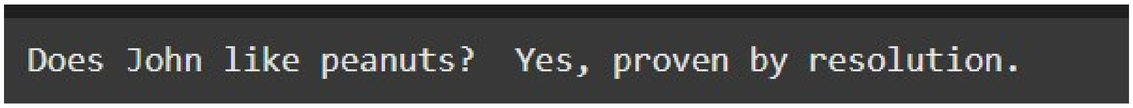
Lab-09 Resolution in First-Order Logic

CODE:

```
from sympy import symbols, And, Or, Not, Implies, to_cnf


# Define constants (entities in the problem)

John, Anil, Harry, Apple, Vegetables, Peanuts, x, y = symbols('John Anil Harry Apple Vegetables
Peanuts x y')


# Define predicates as symbols (this works as a workaround)

Food = symbols('Food')

Eats = symbols('Eats')

Likes = symbols('Likes')

Alive = symbols('Alive')

Killed = symbols('Killed')


# Knowledge Base (Premises) in First-Order Logic

premises = [

  # 1. John likes all kinds of food: Food(x) → Likes(John, x)

  Implies(Food, Likes),


  # 2. Apples and vegetables are food: Food(Apple) ∧ Food(Vegetables)

  And(Food, Food),


  # 3. Anything anyone eats and is not killed is food: (Eats(y, x) ∧ ¬Killed(y)) → Food(x)

  Implies(And(Eats, Not(Killed)), Food),
    # 4. Anil eats peanuts and is still alive: Eats(Anil, Peanuts) ∧ Alive(Anil)
    And(Eats, Alive),

    # 5. Harry eats everything that Anil eats: Eats(Anil, x) → Eats(Harry, x)
    Implies(Eats, Eats),

    # 6. Anyone who is alive implies not killed: Alive(x) → ¬Killed(x)
    Implies(Alive, Not(Killed)),

    # 7. Anyone who is not killed implies alive: ¬Killed(x) → Alive(x)
    Implies(Not(Killed), Alive),
]

# Negated conclusion to prove: ¬Likes(John, Peanuts)
negated_conclusion = Not(Likes)

# Convert all premises and the negated conclusion to Conjunctive Normal Form (CNF)
cnf_clauses = [to_cnf(premise, simplify=True) for premise in premises]
cnf_clauses.append(to_cnf(negated_conclusion, simplify=True))

# Function to resolve two clauses
def resolve(clause1, clause2):
    """
    Resolve two CNF clauses to produce resolvents.
    """
    clause1_literals = clause1.args if isinstance(clause1, Or) else [clause1]
    clause2_literals = clause2.args if isinstance(clause2, Or) else [clause2]
    resolvents = []

    for literal in clause1_literals:
        if Not(literal) in clause2_literals:
            # Remove the literal and its negation and combine the rest
            new_clause = Or(
                *[l for l in clause1_literals if l != literal],
                *[l for l in clause2_literals if l != Not(literal)]
            ).simplify()
            resolvents.append(new_clause)

    return resolvents

# Function to perform resolution on the set of CNF clauses
def resolution(cnf_clauses):
    """
    Perform resolution on CNF clauses to check for a contradiction.
    """
    clauses = set(cnf_clauses)
    new_clauses = set()

    while True:
        clause_list = list(clauses)
        for i in range(len(clause_list)):
            for j in range(i + 1, len(clause_list)):
                resolvents = resolve(clause_list[i], clause_list[j])
                if False in resolvents: # Empty clause found
                    return True # Contradiction found; proof succeeded
                new_clauses.update(resolvents)

        if new_clauses.issubset(clauses): # No new information
            return False # No contradiction; proof failed

        clauses.update(new_clauses)
```

```
# Perform resolution to check if the conclusion follows
result = resolution(cnf_clauses)
print("Does John like peanuts? ", "Yes, proven by resolution." if result else "No, cannot be proven.")
```

OUTPUT:



```
Does John like peanuts?  Yes, proven by resolution.
```