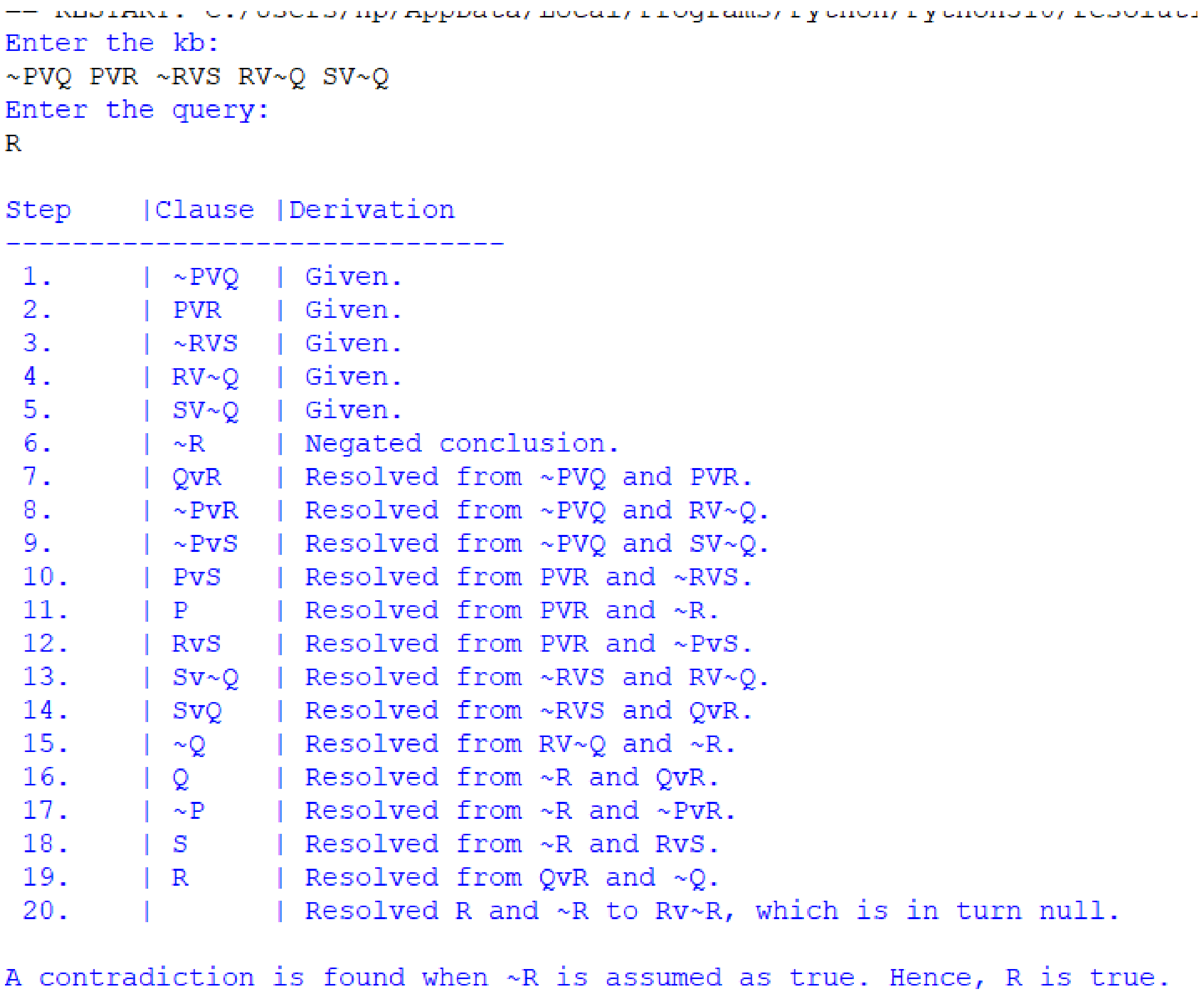
WEEK7

```
import re
def negate(term):
   return f'~{term}' if term[0] != '~' else term[1]


def reverse(clause):
   if len(clause) > 2:
      t = split_terms(clause)
      return f'{t[1]}v{t[0]}'
   return ''
def split_terms(rule):
   exp = '(~*[PQRS])'
   terms = re.findall(exp, rule)
   return terms
def contradiction(query, clause):
   contradictions = [ f'{query}v{negate(query)}', f'{negate(query)}v{query}']
   return clause in contradictions or reverse(clause) in contradictions
def resolve(kb, query):
   temp = kb.copy()
   temp += [negate(query)]
   steps = dict()
   for rule in temp:
      steps[rule] = 'Given.'
   steps[negate(query)] = 'Negated conclusion.'
   i = 0
   while i < len(temp):
      n = len(temp)
      j = (i + 1) % n
      clauses = []
```

```
        while j != i:
            terms1 = split_terms(temp[i])
            terms2 = split_terms(temp[j])
            for c in terms1:
                if negate(c) in terms2:
                    t1 = [t for t in terms1 if t != c]
                    t2 = [t for t in terms2 if t != negate(c)]
                    gen = t1 + t2
                    if len(gen) == 2:
                        if gen[0] != negate(gen[1]):
                            clauses += [f'{gen[0]}v{gen[1]}']
                        else:
                            if contradiction(query,f'{gen[0]}v{gen[1]}'):
                                temp.append(f'{gen[0]}v{gen[1]}')
                                steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn
null. \
                                \nA contradiction is found when {negate(query)} is assumed as true.
Hence, {query} is true."
                                return steps
                    elif len(gen) == 1:
                        clauses += [f'{gen[0]}']
                    else:
                        if contradiction(query,f'{terms1[0]}v{terms2[0]}'):
                            temp.append(f'{terms1[0]}v{terms2[0]}')
                            steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn
null. \
                            \nA contradiction is found when {negate(query)} is assumed as true. Hence,
{query} is true."
                            return steps
            for clause in clauses:
                if clause not in temp and clause != reverse(clause) and reverse(clause) not in temp:
                    temp.append(clause)
                    steps[clause] = f'Resolved from {temp[i]} and {temp[j]}.'
            j = (j + 1) % n
        i += 1
    return steps
def resolution(kb, query):
    kb = kb.split(' ')
    steps = resolve(kb, query)
    print('\nStep\t|Clause\t|Derivation\t')
    print('-' * 30)
    i = 1
    for step in steps:
        print(f' {i}.\t| {step}\t| {steps[step]}\t')
        i += 1
def main():
    print("Enter the kb:")
    kb = input()
    print("Enter the query:")
    query = input()
    resolution(kb,query)
#test 1
#(P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)
main()
#test 2
#(P=>Q)=>Q, (P=>P)=>R, (R=>S)=>~(S=>Q)
main()
```



```
== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python310/resoluti
Enter the kb:
~PVQ PVR ~RVS RV~Q SV~Q
Enter the query:
R

Step    |Clause |Derivation
------------------------------
 1.     | ~PVQ  | Given.
 2.     | PVR   | Given.
 3.     | ~RVS  | Given.
 4.     | RV~Q  | Given.
 5.     | SV~Q  | Given.
 6.     | ~R    | Negated conclusion.
 7.     | QvR   | Resolved from ~PVQ and PVR.
 8.     | ~PvR  | Resolved from ~PVQ and RV~Q.
 9.     | ~PvS  | Resolved from ~PVQ and SV~Q.
 10.    | PvS   | Resolved from PVR and ~RVS.
 11.    | P     | Resolved from PVR and ~R.
 12.    | RvS   | Resolved from PVR and ~PvS.
 13.    | Sv~Q  | Resolved from ~RVS and RV~Q.
 14.    | SvQ   | Resolved from ~RVS and QvR.
 15.    | ~Q    | Resolved from RV~Q and ~R.
 16.    | Q     | Resolved from ~R and QvR.
 17.    | ~P    | Resolved from ~R and ~PvR.
 18.    | S     | Resolved from ~R and RvS.
 19.    | R     | Resolved from QvR and ~Q.
 20.    |       | Resolved R and ~R to Rv~R, which is in turn null.

A contradiction is found when ~R is assumed as true. Hence, R is true.
```