Name : Saakshi Navale

[illegible]

USN : IBM18CS088.

SUB : AI LabTest II

Question: Consider P, Q, and R as variables and the knowledge base contains following sentences.

$Q \Rightarrow R$.

$P \Rightarrow \neg Q$.

$R \vee Q$

Design the code for TT Entailment

```
# checking Entailment.
def checkEntailment ()

    kb = (input (" Enter the knowledge Base"))
    query = (input ("Enter the query"))
    combinations = [ [ True, True, True ],
                     [ True , True, false ],
                     [ True, false, True ],
                     [false, True, True ],
                     [ True, false, false],
                     [ false, false, True],
                     [false, True, false],
                     [false, false, false] ]

    postfix_kb = toPostfix (kb)
    query_kb   = toPostfix (query).
```

1

```
for   combination in combinations:
        eval_kb = evaluatePostfix (postfix_kb, combination)
        eval_q = evaluatePostfix (postfix_query, combination)
        print (combination, ": kb=", eval_kb, ": q=", eval_q)
        if (eval_kb == True):
                if (eval_q == False):
                        print ("doesnot Entail !!")
                        return False

        print (Entails)


# necessary operatfunctions
variable = {'p': 0 , 'q': 1 , 'r': 2}.
priority = {'~': 3 , 'v': 1 , '^', 2}
def _eval (i, val1, val2)
      if  i == '^':
               return val1 and val2
      return val2 and val1 .

def isOperand (c):
      return c.isalpha) and c != 'v'


def is LeftParenthesis (c):
        return  c == '('.
.def is RightParenthesis (c)
        return  c == ')'
```

Laabeli

```
def  isEmpty (stack):
     return    len (stack) == 0

def  peek (stack)
     return    stack[-1]

def    hasLessOrEqualPriority (c1, c2):
       try:
          return priority[c1] <= priority[c2]
       except keyError:
              return false
# converting to postfix
def    toPostFix (Infix)
       stack = []
       postfix = ''
       for   c    in   infix :
             if   isOperand (c):
                  postfix += c
             elsif:  isLeftParenthesis (c):
                     stack.append (c)
             elif    is RightParenthesis (c):
                     operator = stack.pop()
                     while   not   isLeftParenthesis (operator):
                             postfix += operator
                             operator  =  stack.pop()

             else:
                 while (not  isEmpty (stack) and
                             hasLessOrEqualPriority (c, peek(stack)):
                       postfix += stack.pop()
                 stack.append(c)
```

[illegible]

```
while (not isEmpty (stack))
        posfix += stack.pop ()
return postfix ..

# evaluating postfix
def evaluatePostfix (exp, combination):
      stack = []
      for i in exp
          if is operand(i):
              stack.append (comb [variable [i]])
          elif i == '~':
              val1 = stack.pop ()
              stack.append (not val1)
          else :
              val1 = stack.pop()
              val2 = stack.pop()
              stack.append (~~val1 a~~ eval (i, val1, val2))

      return stack.pop()
```

Saakshi