18/11/23 Lab-9 CNF

Aim - To convert the first order logic statement into conjunctive Normal form (CNF).

Code:

```
        import re

def remove brackets (source, id):
    reg = '\((([^\(]* ?\)'
    m = re.search (reg, source)
    if m is None :
        return None, None
    new-source = re. sub(reg, str(id), source, count=1)
    return new-source, m.group(1)

Class logic base :
    def __init__ (self, input):
            self. my-stack = [].
            self. source= input
            final = input
            while 1:
                    input, tmp= remove_brackets (input,
    len (self . my_stack)).
                    if input is None:
                        break
                    final = input
                    self. my_stack . append (tmp)
                self. my-stack . append ( final)

    def get-result (self):
        root = self. my-stack [-1]
        m = re.match ('\s* ([0-9]+) \s * $', root)
        if m is not None:
```

```
        root = self.my_stack[int(m.group(1))]
    reg = '(\d+)'
    while 1:
        m = re.search(reg, root)
        if m is None:
            break
        new = '(' + self.my_stack[int(m.group(1))]
                        + ')'

        root = re.sub(reg, new, root, count=1)
    return root.

def merge_items(self, logic):
    reg0 = '(\d+)'
    reg1 = 'neg\s+(\d+)'
    flag = False.
    for i in range(len(self.my_stack)):
        target = self.my_stack(i)
        if logic not in target:
            continue
        m = re.search(reg1, ~~reg2~~) target)
        if m is not None
            continue.
        m = re.search(reg0, target)
        if m is not None:
            continue.
        for j in re.findall(reg0, target):
            child = self.my_stack[int(j)]
            if logic not in child:
                continue.
            new_reg = "(^|\s)" + j + "(\s|$)"
            self.mystack(i) = re.sub(new_reg, ' '+
child + ' ', self.my_stack(i), count = 1)
            self.my stack(i) = self.my_stack(i).
                                            strip()
```

Date
Page

```
                    flg = True.

    if flg :
            self. merge_items (logic)


class simplification (logic_base):
    def run(self):
        old = self.get_result()
        for i in range (len(self. my_stack )):
            self. my-stack [i]= self. reducing _or(
                                self. my - stack [i])

        final = self. mystack(-1)
        self . my- stack (-1) = self. reducing _and (final)
        return len(old) ! = len (self. get_result())

    def reducing_and(self, target):
        if 'and' not in target :
            return target
        items = set(re.split ('\s+and\s+', target))
        for item in list (items):
            if ('neg' +item) in items:
                return ''
            if re.match ('\d+$', item) is None:
                continue
            value =self. my_stack. count (value)>1:
                value= ''
                self .my-stack [int(item)]= ''
            if value == '':
                items. remove( item)
        return ' and'. join (list (items))

    def reducing_or( self, target):
```

```
if 'or' not in target:
    return target
items = set(re.split('\s+or\s+', target))
for item in list(items):
    if ('neg' + item) in items:
        return ''
return ' or '.join(list(items))


def merging(source):
    old = source.get_result()
    source.merge_items('or')
    source.merge_items('or')
    return old != source.get_result()


def run(input):
    all_strings = []
    zero = ordering(input)
    while zero.run()
        zero = ordering(input)
        while zero.run():
            zero = ordering(zero.get_result())
        merging(zero)
    one = replace_iff(zero.get_result())
    one.run()
    all_strings.append(one.get_result())
    merging(one)
```

output: Enter FOL.

p imp q.

steps:

p im q

neg p or q.