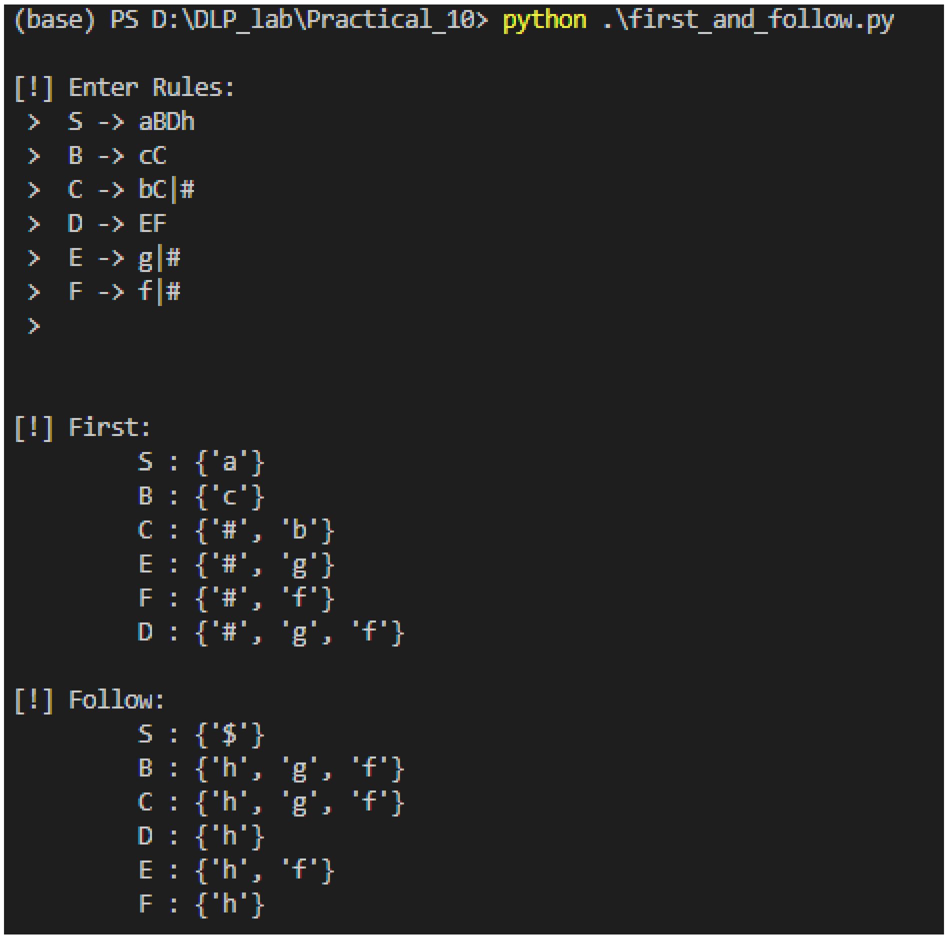
Practical: 10

Aim: Write a program that take grammar as input and find the FIRST SET and FOLLOW SET of all nonterminal symbols of grammar.

Program:

```
from collections import defaultdict
def isNonTerminal(x):
    return x.isupper()

def isTerminal(x):
    return not isNonTerminal(x)

def parse(s):
    s = s.replace(' ', '')
    s = s.split('->')
    if len(s) != 2:
        print(s)
        raise Exception()

    lhs, rhs = s

    lhs = lhs.strip()
    rhs = rhs.strip()

    if len(lhs) == 0 or len(rhs) == 0:
        raise Exception()

    rhsParts = [part.strip() for part in rhs.split('|')]
    if '' in rhsParts:
        rhsParts.remove('')

    return lhs, rhsParts


def readRules():

    rules = []
    print('[!] Enter Rules:')

    while True:
        try:
            s = input(' >  ').strip()
        except EOFError:
            break

        if s == '':
            break
        rules += [s]

    return rules


def getFirst(rules):


    # nullables ----------------------
    tmp = []
    nullables = []

    for rule in reversed(rules):
        lhs, rhsParts = parse(rule)

        for part in rhsParts:
            if part == '#':
                nullables += [lhs]
                break

    for rule in reversed(rules):
        lhs, rhsParts = parse(rule)

        for part in rhsParts:
            for c in part:
                if isNonTerminal(c) and c in nullables:
                    part = part.replace(c, '#')
            part = part.replace('#', '')
            if part == '':
                nullables += [lhs]
                break

    nullables = list(set(nullables))
    # ---------------------------------



    allNonTerminals = []

    for rule in rules:
        lhs, _ = parse(rule)
        allNonTerminals += lhs

    allNonTerminals = list(set(allNonTerminals))

    queue = rules.copy()
    found = defaultdict(bool)
    firstSet = defaultdict(set)
    while len(queue) != 0:
        rule = queue[0]
        lhs, rhsParts = parse(rule)
        if found[lhs]:
            queue.remove(rule)
            continue

        isFirstNonTerminal = False

        for part in rhsParts:
            if isTerminal(part[0]):
                continue

            if found[part[0]]:
                continue

            isFirstNonTerminal = True

        if isFirstNonTerminal:
            queue.remove(rule)
            queue.append(rule)
            continue

        # Current rule only contains terminals
        currentFirst = set()
        for rhsPart in rhsParts:
            if isTerminal(rhsPart[0]):
                currentFirst.add(rhsPart[0])
            else:
                for c in rhsPart:
                    currentFirst = currentFirst | firstSet[c]
                    if c not in nullables:
                        break

        firstSet[lhs] = currentFirst
        found[lhs] = True

    return firstSet




def getFollow(rules, first):
    allNonTerminals = set()

    # nullables -----------------------
    tmp = []
    nullables = []
    for rule in reversed(rules):
        lhs, rhsParts = parse(rule)

        for part in rhsParts:
            if part == '#':
                nullables += [lhs]
                break

    for rule in reversed(rules):
        lhs, rhsParts = parse(rule)

        for part in rhsParts:
            for c in part:
                if isNonTerminal(c) and c in nullables:
                    part = part.replace(c, '#')
            part = part.replace('#', '')
            if part == '':
                nullables += [lhs]
                break

    nullables = list(set(nullables))
    # ---------------------------------


    followSet = defaultdict(set)

    for rule in rules:
        lhs, rhsParts = parse(rule)
        allNonTerminals.add(lhs)

    for tmprule in rules:
        nonTerminal = parse(tmprule)[0]

        if nonTerminal == START:
            followSet[nonTerminal] = set('$')
            continue

        for rule in rules:
            if nonTerminal not in rule.split('->')[1]:
                continue

            lhs, rhsParts = parse(rule)

            for part in rhsParts:
                if nonTerminal not in part:
                    continue
                index = part.index(nonTerminal)
                nextStr = part[index+1:]
                if nextStr == '':
                    if lhs != nonTerminal:
                        newSet = followSet[lhs] | followSet[nonTerminal]
                        followSet[lhs] = newSet
                        followSet[nonTerminal] = newSet
                counter = 0

                for c in nextStr:
                    if isTerminal(c):
                        followSet[nonTerminal].add(c)
                        counter += 1
                        break

                    for x in first[c]:
                        if x != '#':
                            followSet[nonTerminal].add(x)


                    if c not in nullables:
                        counter += 1
                        break

                if counter == 0:
                    for c in followSet[lhs]:
                        followSet[nonTerminal].add(c)

    return followSet


def getFirstFollow(rules):
    first = getFirst(rules)
    follow = getFollow(rules, first)

    print('\n[!] First:')
    printdd(first)

    print('\n[!] Follow:')
    printdd(follow)

    return first, follow


def getNonTerminals(rules):
    x = set()

    for rule in rules:
        lhs, rhsParts = parse(rule)
        x.add(lhs)
    return x

def getTerminals(rules):
    x = set()

    for rule in rules:
        lhs, rhsParts = parse(rule)
        for part in rhsParts:
            for c in part:
                if isTerminal(c):
                    x.add(c)
    return x

def getStart(rules):
    for x in rules:
        lhs, _ = parse(x)
        if lhs == START:
            return x
    raise Exception()

def getRhs(rules, lhs):
    for x in rules:
        l, _ = parse(x)
        if lhs == l:
            return x
    return ''

def printdd(x):
    for k in x:
        print('\t', k, ':',  x[k])

def main():
    rules = readRules()

    print()
    first, follow = getFirstFollow(rules)
    print()

print()
main()

'''
S -> aBDh
B -> cC
C -> bC|#
D -> EF
E -> g|#
F -> f|#
'''
```

Output:



```
(base) PS D:\DLP_lab\Practical_10> python .\first_and_follow.py

[!] Enter Rules:
 >  S -> aBDh
 >  B -> cC
 >  C -> bC|#
 >  D -> EF
 >  E -> g|#
 >  F -> f|#
 >


[!] First:
         S : {'a'}
         B : {'c'}
         C : {'#', 'b'}
         E : {'#', 'g'}
         F : {'#', 'f'}
         D : {'#', 'g', 'f'}

[!] Follow:
         S : {'$'}
         B : {'h', 'g', 'f'}
         C : {'h', 'g', 'f'}
         D : {'h'}
         E : {'h', 'f'}
         F : {'h'}
```

Conclusion: From this practical I have learnt about how to implement code for finding first and follow set of any non-terminal for given grammar.