**ELIMINATING LEFT RECURSION AND LEFT FACTORING**

Code-Left Recursion

n = int(input("Enter number of grammer"))

while n:

  s= input("Enter the grammar:")

# grammar format S->Sab/xy

  if s[0]!=s[3]:

    print("No left recursion")

  if s[0]==s[3]:

    x=s[0]

    l=len(s)

    a=''

    c=''

    b=s[0]+"'"

    #print(b)

    for i in range(l):

        if s[i]=="/":

            a=a+s[i+1:]

    for i in range(l):

        if s[i]=="/":

            c=c+s[4:i]

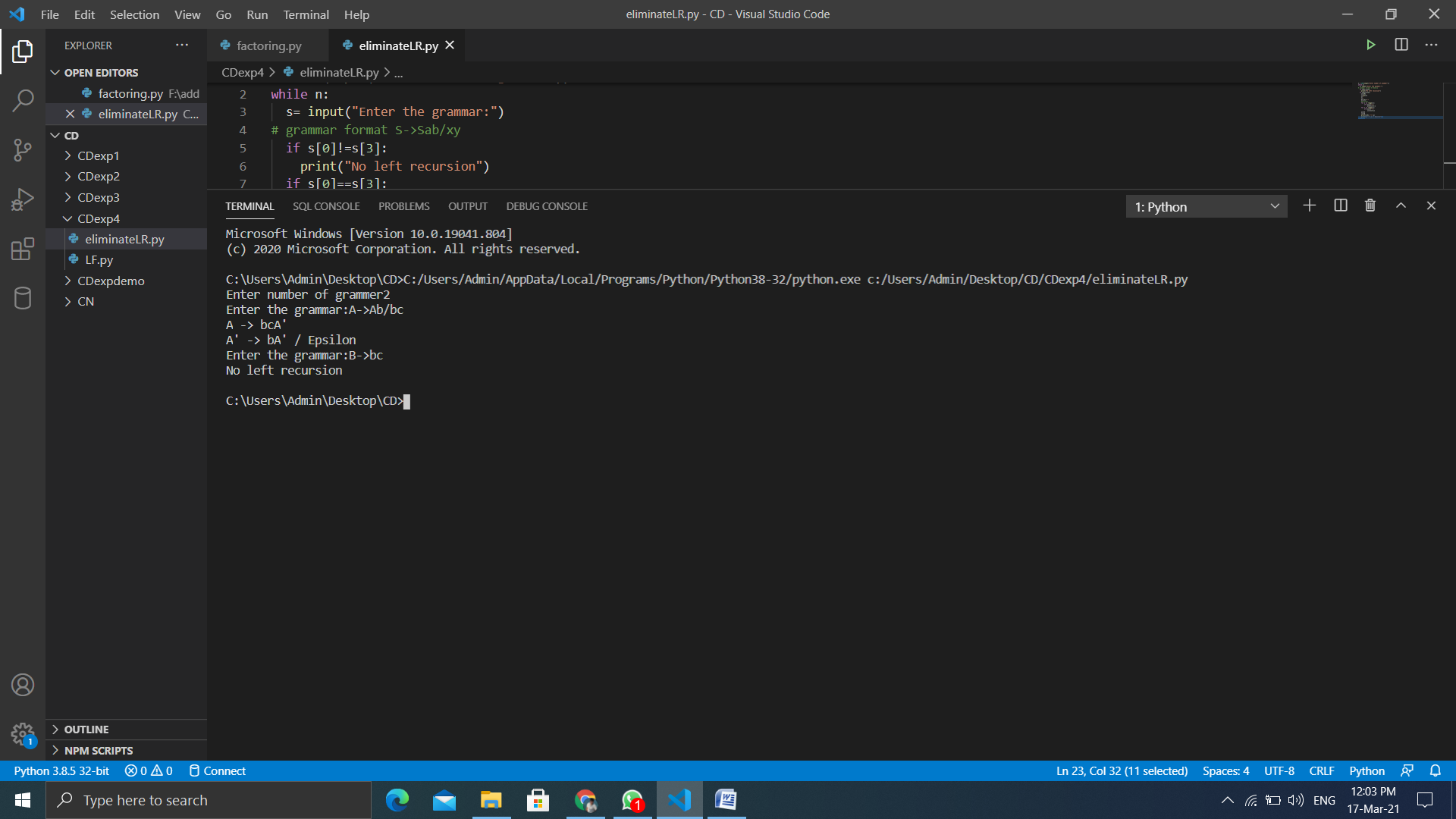
    a=a+b

    c=c+b

    print(s[0],"->",a)

    print(b,"->",c,"/ Epsilon")

  n = n-1

Output:****

Code-Left Factoring

import sys

import re

import time

sys.setrecursionlimit(60)

def isTerm(ter):

    if ter == '#':

        return True

    for t in terms:

        if ter == t:

            return True

    return False

def isNTerm(nT):

    for nt in nonterms:

        if nT == nt:

            return True

    return False

def LeftCheck(nT, search, escape):

    # print("NT:-"+nT+",Search:-"+search)

    # time.sleep(3)

    for prod in production\_dict[nT]:

        if search == prod[0]:

            leftcheck\_list[search].append(nT)

            return True

        else:

            if isNTerm(prod[0]):

                return LeftCheck(prod[0], search, escape+1)

            else:

                escape = 0

                continue

    return False

def RightCheck(nT, search):

    # print("NT:-"+nT+",Search:-"+search)

    for prod in production\_dict[nT]:

        if search == prod[len(prod)-1]:  # Direct

            return True

    return False

def Left\_toRight(nT):

    bet = []

    alpha = []

    #expand on indirect relations

    for nont in leftcheck\_list[nt]:

        if nont != nT:

            for i in range(0, len(production\_dict[nT])):

                prod = production\_dict[nT][i]

                if(nont == prod[0]):

                    prod\_temp = prod

                    remain = prod[1:]

                    for p in production\_dict[nont]:

                        if nT == p[0]:

                            final\_prod = p + remain

                            production\_dict[nT].pop(i)

                            production\_dict[nT].insert(i, final\_prod)

    #remove Direct Left recursion

    for prd in production\_dict[nT]:

        if nT == prd[0]:

            alpha.append(prd[1:])

        else:

            bet.append(prd)

    gram\_dash = ""

    for a in alpha:

        gram\_dash = gram\_dash + a + nT + '\'' + "/"

    gram\_dash = gram\_dash[0:-1]

    gram = ""

    for b in bet:

        gram = gram+b+nT+'\''+"/"

    gram = gram[0:-1]

    print(nT+"->"+gram)

    print(nT+"\'->"+gram\_dash+"/#")

def Prod\_print(nT):

    full\_prod = ""

    for pd in production\_dict[nT]:

        full\_prod = full\_prod+pd+"/"

    full\_prod = full\_prod[0:-1]

    print(nT+"->"+full\_prod)

productions = []

n = input("Enter the number of Productions:-")

n = int(n)

print("\nRules:\n--------------------------------------------------------------------------------------\nEpsilon is represented by  # \nProductions are of the form A->B, where ‘A’ is a single Non-Terminal and ‘B’ can be any combination of Terminals and Non - Terminals.\nTerminals with only single characters work\nDO NOT use the same char for terminal and non terminal.\nDo not use # or $ as they are reserved for special purposes.\n\n")

for i in range(n):

    prod = input()

    prod.strip()

    productions.append(prod)

nonterms = []

terms = []

#Since Productions are context free grammaer there is only one nonterminal on the left and then an arrow so we can check for terms after the arrow

#Finding Non Terminals:-

for i in range(n):

    nonterms.append(productions[i][0])

#Finding Terminals:-

for i in productions:

     for j in range(3, len(i)):

          check = True

          for nt in nonterms:

              if i[j] == nt or i[j] == '#' or i[j] == '/':

                  check = False

          if check:

              terms.append(i[j])

print("Non Terminals:-", nonterms)

print("Terminals:-", terms)

print("Productions:-", productions)

#Production Dict

production\_dict = {}

for nt in nonterms:

    production\_dict[nt] = []

# split the productions into parts to simplify parsing

for production in productions:

    nonterminal\_to\_production = production.split("->")

    expanded = nonterminal\_to\_production[1].split(

        "/")  # assumption : single char terminals

    for ex in expanded:

        production\_dict[nonterminal\_to\_production[0]].append(ex)

print("production\_dict", production\_dict)

leftcheck\_dict = {}

leftcheck\_list = {}  # Stores the non terminals which have indirect left recursion

for nt in nonterms:

    leftcheck\_list[nt] = []

    leftcheck\_dict[nt] = LeftCheck(nt, nt, 0)

rightcheck\_dict = {}

for nt in nonterms:

    rightcheck\_dict[nt] = RightCheck(nt, nt)

print("\n")

for nt in nonterms:

    if leftcheck\_dict[nt]:

        print(nt+" contains Left Recursive Grammar")

    if rightcheck\_dict[nt]:

        print(nt+" contains Right Recursive Grammar")

print("--------------------------------\nLeft Eliminated Grammer:-")

for nt in nonterms:

    if leftcheck\_dict[nt] == True:

        Left\_toRight(nt)

    else:

        Prod\_print(nt)

Output:

