**Practical 1 – Construct NDFA**

**CODE:**

#Install package 'automata-lib'

#using the following command:

#pip (or pip3) install automata-lib

from automata.fa.nfa import NFA

class NDFA:

def \_\_init\_\_(self):

state\_set = set(input("Enter state set>\t"))

input\_symbols = set(input("Enter input symbol set>\t"))

initial\_state = input("Enter the initial state>\t")

final\_states = set(input("Enter the final state(s)>\t"))

rule\_count = int(input("Enter the number of rules you want to add>\t"))

rules = []

for counter in range(rule\_count):

rules.append(input("Enter rule " + str(counter + 1) + ">\t").replace(" ",""))

rules = self.get\_transitions(rules)

self.nfa = NFA(

states = state\_set,

input\_symbols = input\_symbols,

transitions = rules,

initial\_state = initial\_state,

final\_states = final\_states

)

del state\_set, input\_symbols, initial\_state, final\_states, rules

def get\_transitions(self, rules):

rules = [i.split("->") for i in rules]

rules\_dict = {}

for rule in rules:

if rule[0] not in rules\_dict:

rules\_dict[rule[0]] = {rule[1][0]:rule[1][1]}

else:

rules\_dict[rule[0]][rule[1][0]] = rule[1][1]

return rules\_dict

def print\_stats(self):

print("\n\nSet of states are > ", self.nfa.states)

print("Input symbols are > ", self.nfa.input\_symbols)

print("Transitions are > ")

for transition in self.nfa.transitions:

print(transition, self.nfa.transitions[transition])

print("Initial state > ", self.nfa.initial\_state)

print("Final states > ", self.nfa.final\_states)

def print\_transition\_table(self):

input\_symbols = list(self.nfa.input\_symbols)

transitions = self.nfa.transitions

print("\n\nTransitions Table is > ")

print("States\t\t" + str(input\_symbols[0]) + "\t\t" + str(input\_symbols[1]))

for transition in transitions:

for input\_symbol in input\_symbols:

try:

temp = transitions[transition][input\_symbol]

del temp

except KeyError:

transitions[transition][input\_symbol] = "-"

print(transition + "\t\t" + transitions[transition][input\_symbols[0]] + "\t\t" + transitions[transition][input\_symbols[1]])

del input\_symbols, transitions

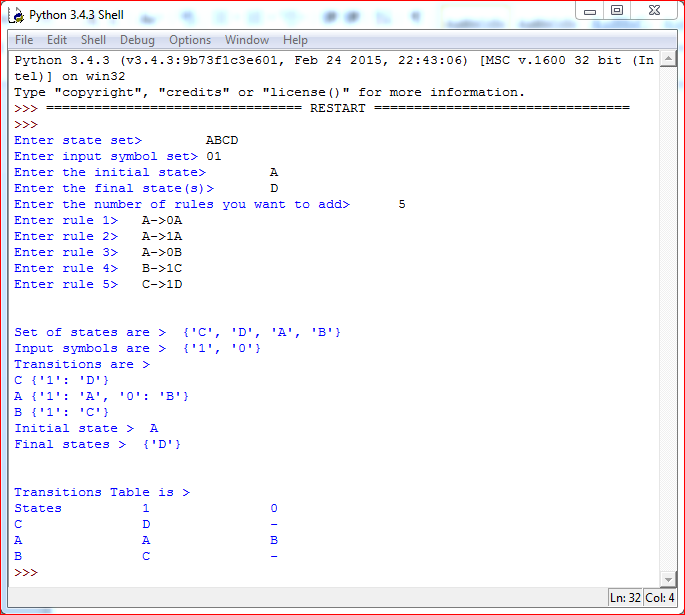
if \_\_name\_\_ == "\_\_main\_\_":

ndfa = NDFA()

ndfa.print\_stats()

ndfa.print\_transition\_table()

**OUTPUT:**



**Practical 2 – Convert Right Linear to Left Linear**

**CODE:**

def get\_transitions(rules):

my\_dict = {}

ld = ''

res = dict()

r = ''

for i in rules:

if i[0] not in my\_dict:

my\_dict[i[0]] = [i[1][1], i[1][0]]

else:

my\_dict[i[0]][i[1][0]] = i[1][1]

for sub in my\_dict:

if isinstance(my\_dict[sub], list):

res[sub] = ld.join([str(ele) for ele in my\_dict[sub]])

print("The Left Linear Grammar")

for item in res:

r = item + "-" + str(res[item]) + "\n"

print(str(r))

rule\_count = int(input("Enter number of rules >\t"))

rules = []

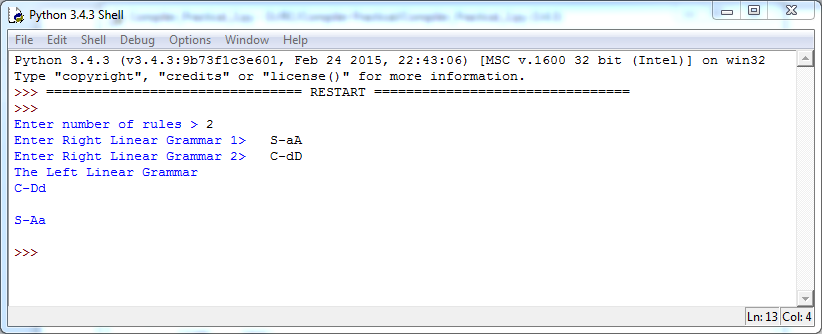
for i in range(rule\_count):

rules.append(input("Enter Right Linear Grammar "+ str(i+1) +">\t"))

rules = [i.split("-") for i in rules]

get\_transitions(rules)

**OUTPUT:**



**Practical 3 – DAG**

**CODE:**

def func1(x):

main = []

for i in range(0,x):

y = input()

main.append(y)

print("Label Operator Left Right")

for i in range(0,x):

q = main[i]

if q[0] not in res:

res.append(q[0])

if(len(q)>3):

print("(", q[0] ,")", " ", q[3], " ", q[2], " ",q[4])

else:

print("(", q[0] ,")", " ", q[1], " ", q[2], " ")

print(main)

print(res)

print("Enter number of 3 - Address Code: ")

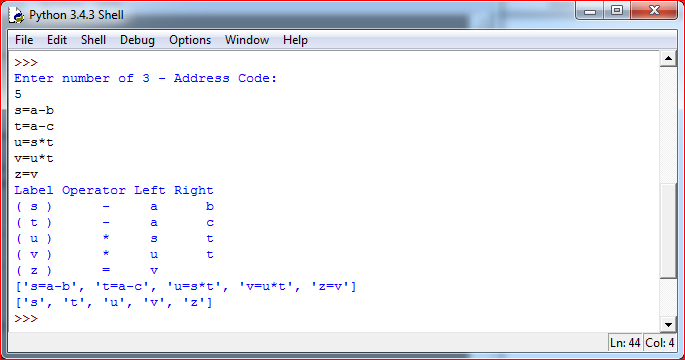
x = input()

x = int(x)

res = []

func1(x)

**OUTPUT:**



**Practical 4 – Triples**

**CODE:**

def func1(x):

main = []

for i in range(0,x):

y = input()

main.append(y)

print("Address Operator Argument1 Argument2")

for i in range(0,x):

q = main[i]

if q[0] not in res:

res.append(q[0])

e = func2(q[2])

if(len(q)>3):

r = func2(q[4])

print("(", i ,")", " ", q[3], " ", e, " ",r)

else:

print("(", i ,")", " ", q[1], " ", e, " ")

print(main)

print(res)

def func2(q):

try:

z = res.index(q)

return(z)

except:

return(q)

print("Enter number of production: ")

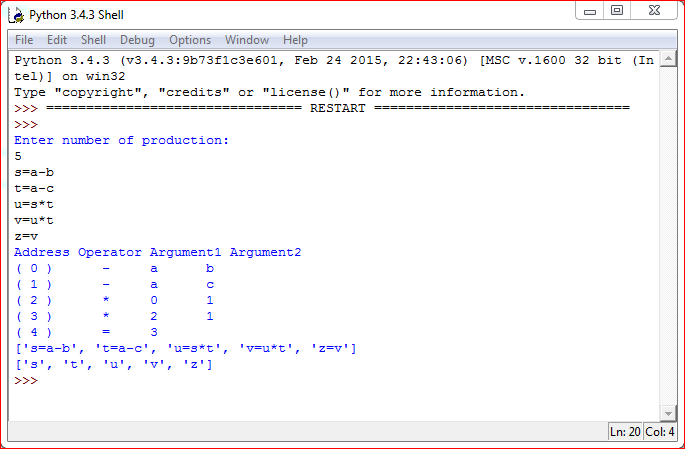
x = input()

x = int(x)

res = []

func1(x)

**OUTPUT:**



**Practical 5 – Postfix Evaluation**

**CODE:**

def postfix\_evaluation(s):

s = s.split()

n = len(s)

stack = []

for i in range(n):

if s[i].isdigit():

stack.append(int(s[i]))

elif s[i] == "+":

a = stack.pop()

b = stack.pop()

stack.append(int(a) + int(b))

elif s[i] == "\*":

a = stack.pop()

b = stack.pop()

stack.append(int(a) \* int(b))

elif s[i] == "/":

a = stack.pop()

b = stack.pop()

stack.append(int(a) / int(b))

elif s[i] == "-":

a = stack.pop()

b = stack.pop()

stack.append(int(a) - int(b))

return stack.pop()

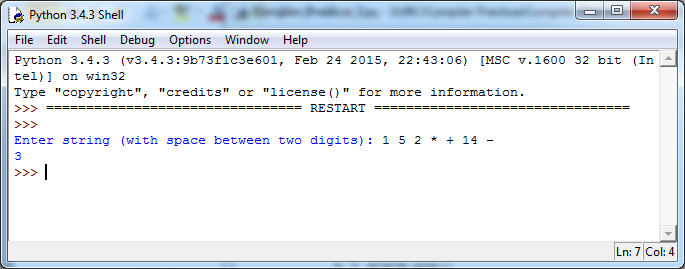
s = input("Enter string (with space between two digits): ")

#s = "1 5 2 \* + 14 -"

val = postfix\_evaluation(s)

print(val)

**OUTPUT:**



**Practical 6 – 3 Address Code**

**CODE:**

postfix = input("Enter Postfix Expression: ").split()

operators = ['+','-','/','\*','^']

stack = []

result = ' '

str1 = ' '

count = 0

print("3 address code")

for i in postfix:

if i not in operators:

stack.append(i)

print("stack=", stack)

else:

op1 = stack.pop()

op2 = stack.pop()

result = op2 + i +op1

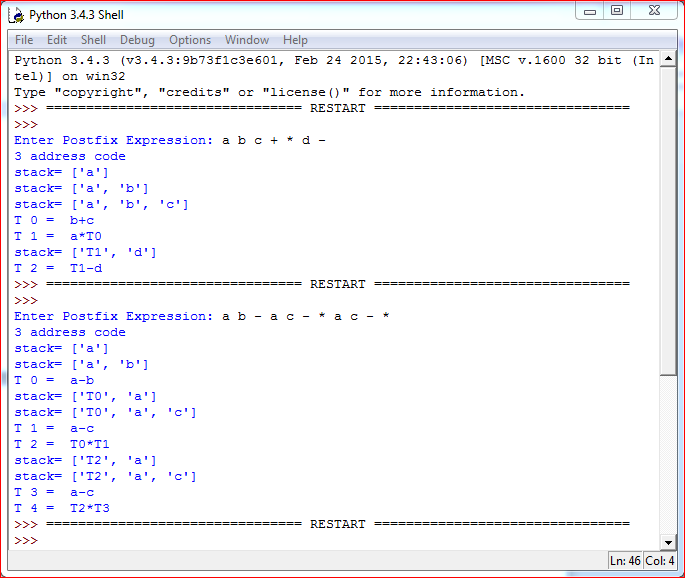
str1 = 'T' + str(count)

stack.append(str1)

print("T", count, "= ", result)

count += 1

**OUTPUT:**



**Practical 7 – Loop Jamming**

**CODE:**

import time

from datetime import datetime

def func(arr1, arr2, arr3):

t1=datetime.now()

start = time.time()

print(t1.minute, " : ", t1.second, " : ", t1.microsecond)

for i in range (0, 10000000):

sum=0

for j in range (0, len(arr1)):

sum += arr1[j]

for k in range (0, len(arr2)):

sum += arr2[k]

for l in range (0, len(arr3)):

sum += arr3[l]

if(sum != 210):

print(false)

tm=datetime.now()

done = time.time()

elapsed = done - start

print(tm.minute, " : ", tm.second, " : ", tm.microsecond)

print("First Loop Difference: ", elapsed)

start = time.time()

for i in range (0, 10000000):

sum = 0

for j in range (0, len(arr1)):

sum += arr1[j]

sum += arr2[j]

sum += arr3[j]

if(sum != 210):

print(false)

tn=datetime.now()

done = time.time()

elapsed = done - start

print(tn.minute, " : ", tn.second, " : ", tn.microsecond)

print("Second Loop Difference: ", elapsed)

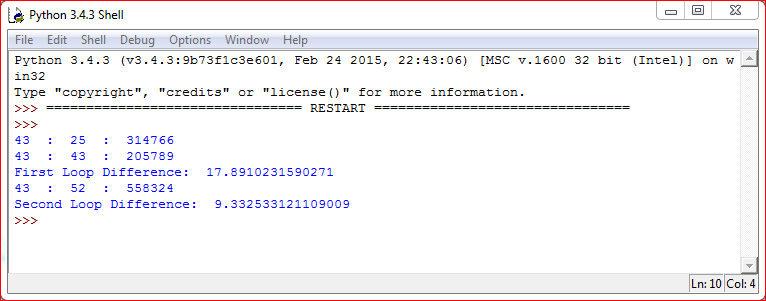
arr1 = [10,20,30]

arr2 = [20,10,30]

arr3 = [40,40,10]

func(arr1, arr2, arr3)

**OUTPUT:**



**Practical 8 – Loop Unrolling**

**CODE:**

import time

from datetime import datetime

def func\_LoopUnrolling():

arr = []

arr1 = []

t1=datetime.now()

start = t1.microsecond

print(start)

for i in range (0, 1000):

arr.insert(0, i)

print(arr)

t2=datetime.now()

end1 = t2.microsecond

print(end1)

for i in range (0, 1000, 4):

arr1.insert(0, i)

arr1.insert(0, i + 1)

arr1.insert(0, i + 2)

arr1.insert(0, i + 3)

print(arr1)

t3=datetime.now()

end2 = t3.microsecond

print(end2)

print("Before Unrolling: ", end1 - start)

print("After Unrolling: ", end2 - end1)

func\_LoopUnrolling()

**OUTPUT:**

