

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
```

Compiler Practical

)

```
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)
```

Compiler Practical

```
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:

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter state set>          ABCD
Enter input symbol set> 01
Enter the initial state>      A
Enter the final state(s)>      D
Enter the number of rules you want to add>      5
Enter rule 1>      A->0A
Enter rule 2>      A->1A
Enter rule 3>      A->0B
Enter rule 4>      B->1C
Enter rule 5>      C->1D

Set of states are > {'C', 'D', 'A', 'B'}
Input symbols are > {'1', '0'}
Transitions are >
C {'1': 'D'}
A {'1': 'A', '0': 'B'}
B {'1': 'C'}
Initial state > A
Final states > {'D'}

Transitions Table is >
States      1      0
C           D      -
A           A      B
B           C      -
>>>
Ln: 32 Col: 4

```

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 = []
```

Compiler Practical

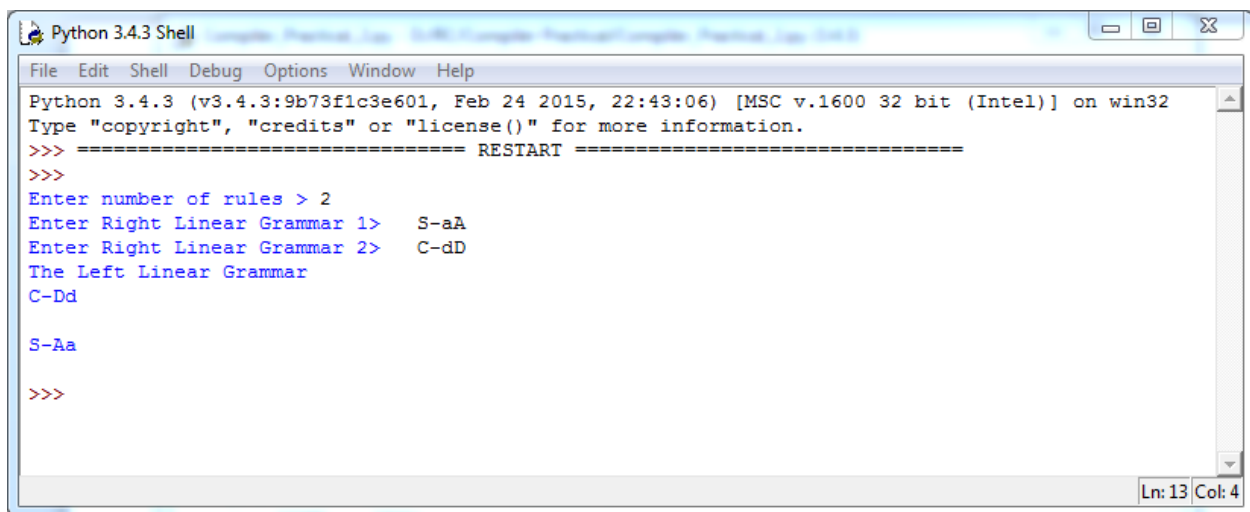
```
for i in range(rule_count):
```

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

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

```
get_transitions(rules)
```

OUTPUT:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter number of rules > 2
Enter Right Linear Grammar 1> S-aA
Enter Right Linear Grammar 2> C-dD
The Left Linear Grammar
C-Dd

S-Aa

>>>
```

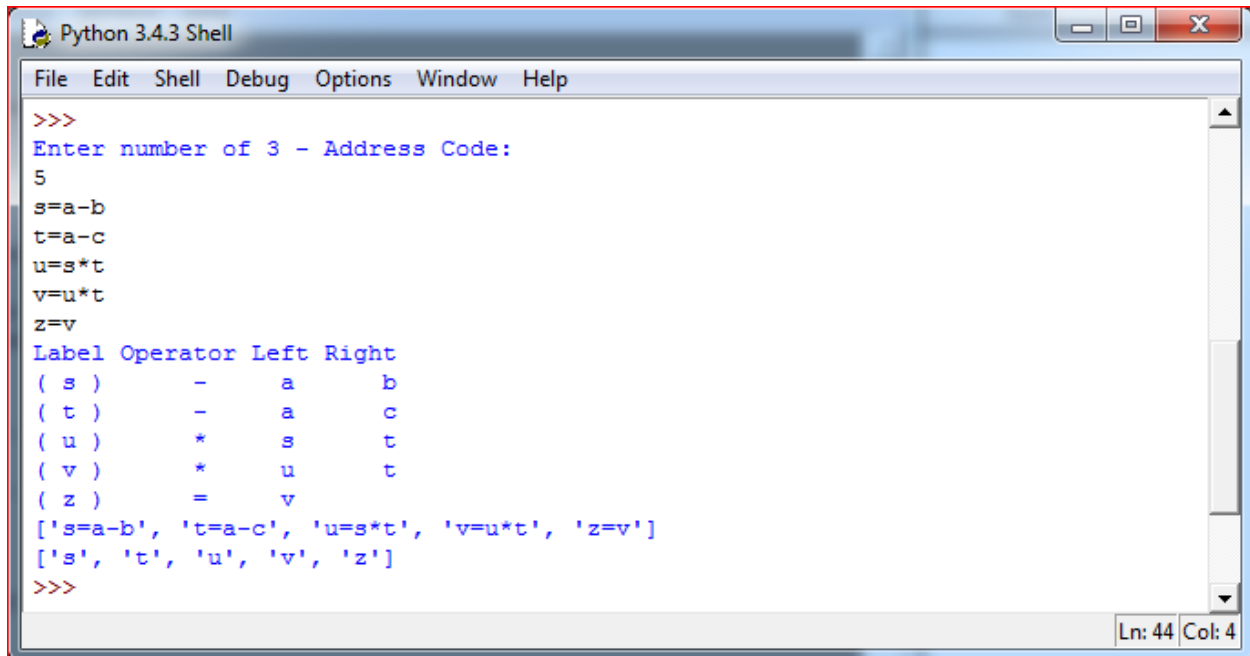
Ln: 13 Col: 4

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:



The screenshot shows a Python 3.4.3 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a text area containing the following code:

```
>>>
Enter number of 3 - Address Code:
5
s=a-b
t=a-c
u=s*t
v=u*t
z=v
Label Operator Left Right
( s )      -      a      b
( t )      -      a      c
( u )      *      s      t
( v )      *      u      t
( z )      =      v
['s=a-b', 't=a-c', 'u=s*t', 'v=u*t', 'z=v']
['s', 't', 'u', 'v', 'z']
>>>
```

The status bar at the bottom right indicates "Ln: 44 Col: 4".

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)
```

Compiler Practical

```
    return(z)
```

```
except:
```

```
    return(q)
```

```
print("Enter number of production: ")
```

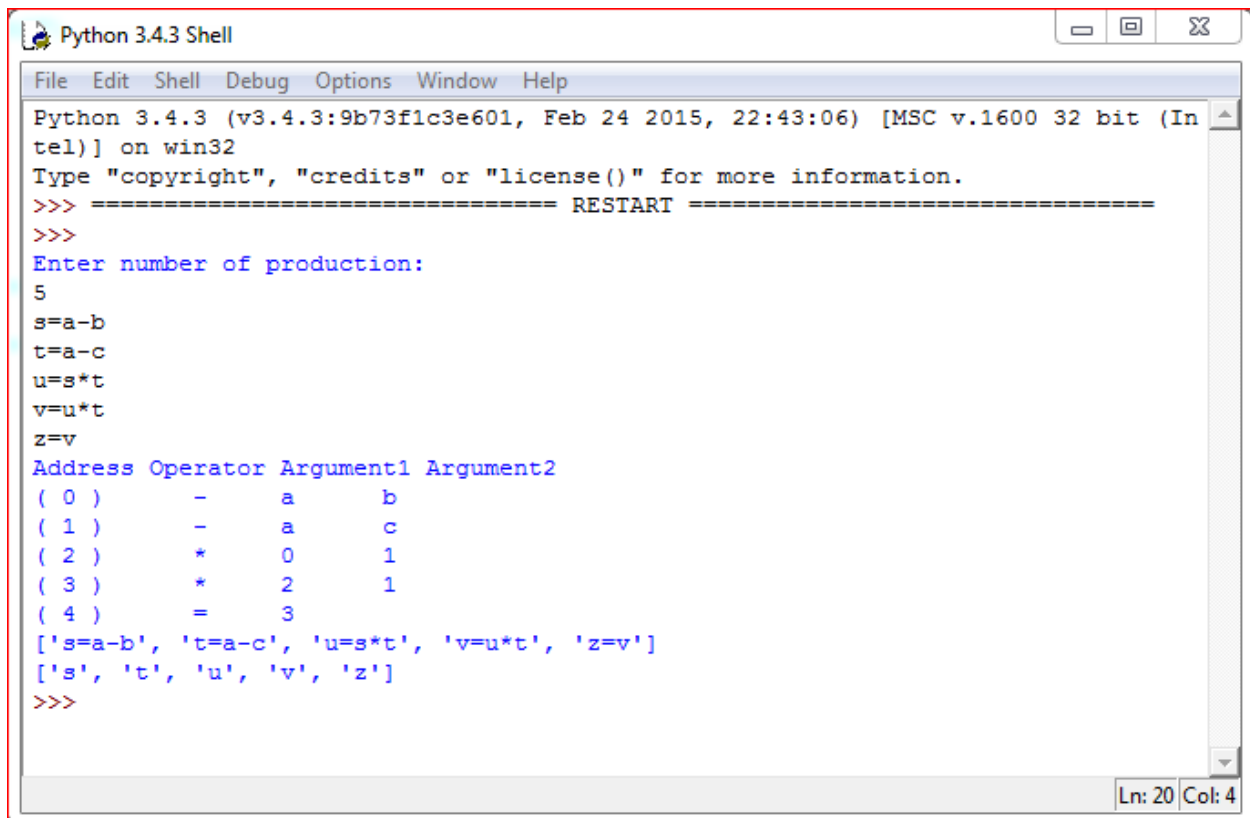
```
x = input()
```

```
x = int(x)
```

```
res = []
```

```
func1(x)
```

OUTPUT:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter number of production:
5
s=a-b
t=a-c
u=s*t
v=u*t
z=v
Address Operator Argument1 Argument2
( 0 )      -      a      b
( 1 )      -      a      c
( 2 )      *      0      1
( 3 )      *      2      1
( 4 )      =      3
['s=a-b', 't=a-c', 'u=s*t', 'v=u*t', 'z=v']
['s', 't', 'u', 'v', 'z']
>>>
```

Ln: 20 Col: 4

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))
```

Compiler Practical

```
return stack.pop()
```

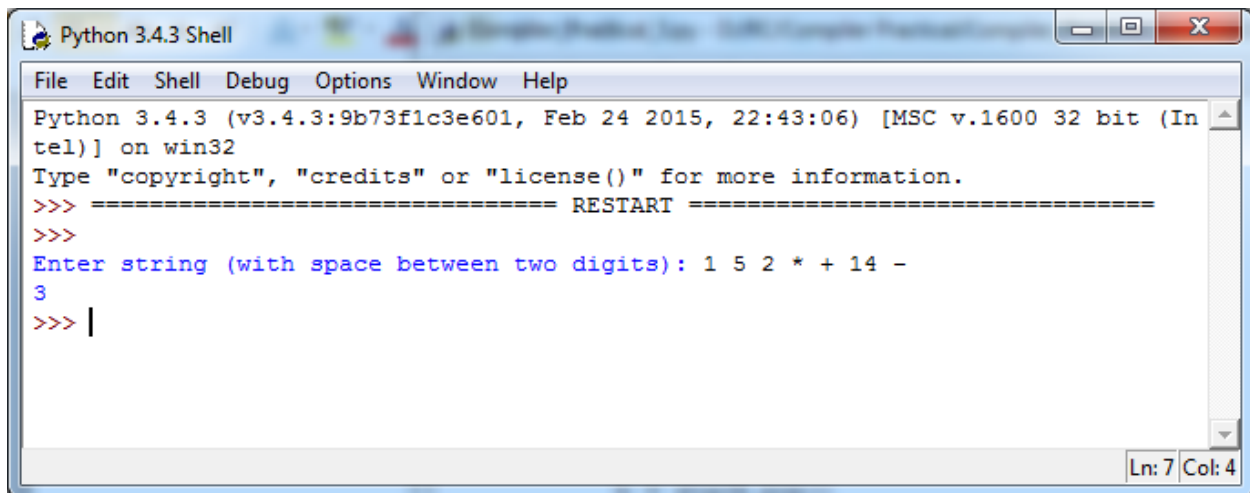
```
s = input("Enter string (with space between two digits): ")
```

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

```
val = postfix_evaluation(s)
```

```
print(val)
```

OUTPUT:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter string (with space between two digits): 1 5 2 * + 14 -
3
>>> |
```

Ln: 7 Col: 4

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
```

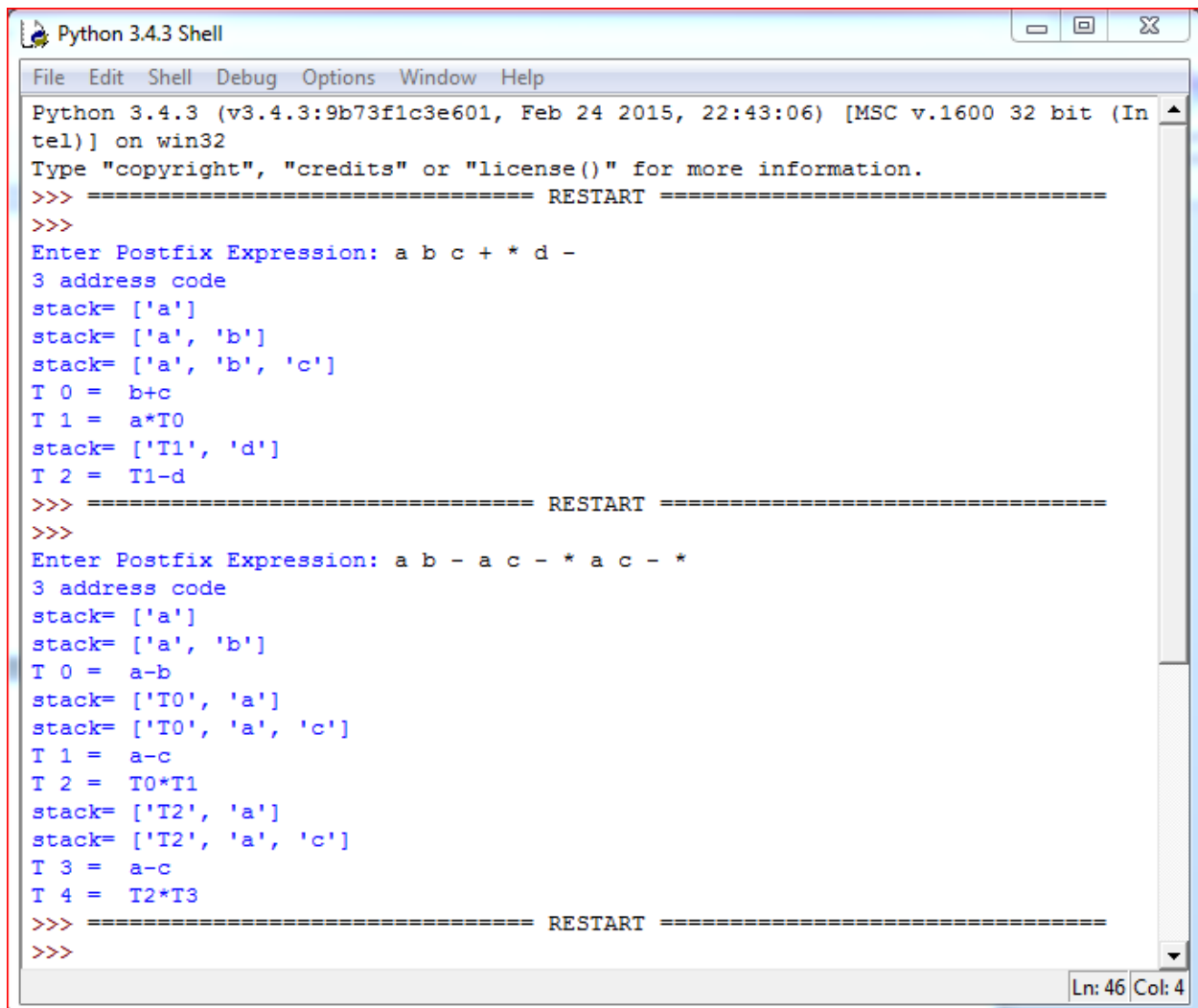
```
        str1 = 'T' + str(count)
```

```
        stack.append(str1)
```

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

```
        count += 1
```

OUTPUT:



The screenshot shows a Python 3.4.3 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a status bar (Ln: 46 Col: 4). The window displays the output of a postfix expression evaluator. It starts with a version and build information message, followed by a prompt for copyright information. The user enters a postfix expression, and the program evaluates it step-by-step, showing the stack and temporary variables (T0, T1, T2) at each step. The process is repeated for two more expressions.

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter Postfix Expression: a b c + * d -
3 address code
stack= ['a']
stack= ['a', 'b']
stack= ['a', 'b', 'c']
T 0 = b+c
T 1 = a*T0
stack= ['T1', 'd']
T 2 = T1-d
>>> ===== RESTART =====
>>>
Enter Postfix Expression: a b - a c - * a c - *
3 address code
stack= ['a']
stack= ['a', 'b']
T 0 = a-b
stack= ['T0', 'a']
stack= ['T0', 'a', 'c']
T 1 = a-c
T 2 = T0*T1
stack= ['T2', 'a']
stack= ['T2', 'a', 'c']
T 3 = a-c
T 4 = T2*T3
>>> ===== RESTART =====
>>>
```

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)
```

Compiler Practical

```
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]
```

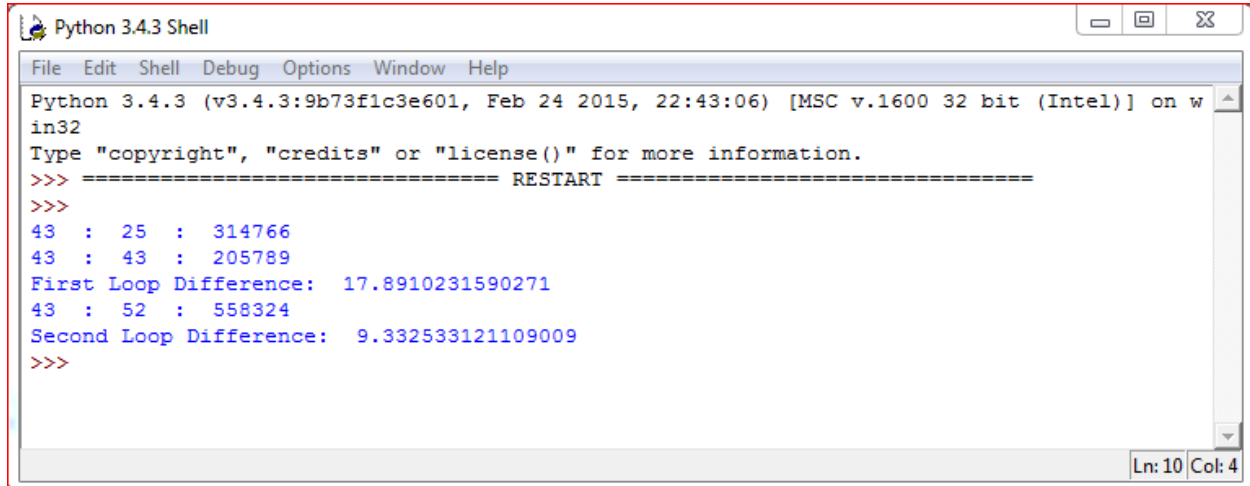
```
arr2 = [20,10,30]
```

```
arr3 = [40,40,10]
```

```
func(arr1, arr2, arr3)
```


Compiler Practical

OUTPUT:

A screenshot of a Python 3.4.3 Shell window. The window has a title bar that says "Python 3.4.3 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
43 : 25 : 314766
43 : 43 : 205789
First Loop Difference: 17.8910231590271
43 : 52 : 558324
Second Loop Difference: 9.332533121109009
>>>
```

The status bar at the bottom right indicates "Ln: 10 Col: 4".

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)
```

Compiler Practical

```
t3=datetime.now()
```

```
end2 = t3.microsecond
```

```
print(end2)
```

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

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

func_LoopUnrolling()

OUTPUT:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f10c601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART =====
>>>
30 : 58 : 482241
1648436550.5134441
[999, 998, 997, 996, 995, 994, 993, 992, 991, 990, 989, 988, 987, 986, 985, 984, 983, 982, 981, 980, 979, 978, 977, 976, 975, 974, 973, 972, 971, 970, 969, 968, 967, 9
66, 965, 964, 963, 962, 961, 960, 959, 958, 957, 956, 955, 954, 953, 952, 951, 950, 949, 948, 947, 946, 945, 944, 943, 942, 941, 940, 939, 938, 937, 936, 935, 934, 933
, 932, 931, 930, 929, 928, 927, 926, 925, 924, 923, 922, 921, 920, 919, 918, 917, 916, 915, 914, 913, 912, 911, 910, 909, 908, 907, 906, 905, 904, 903, 902, 901, 900,
, 899, 898, 897, 896, 895, 894, 893, 892, 891, 890, 889, 888, 887, 886, 885, 884, 883, 882, 881, 880, 879, 878, 877, 876, 875, 874, 873, 872, 871, 870, 869, 868, 867, 86
6, 865, 864, 863, 862, 861, 860, 859, 858, 857, 856, 855, 854, 853, 852, 851, 850, 849, 848, 847, 846, 845, 844, 843, 842, 841, 840, 839, 838, 837, 836, 835, 834, 833,
832, 831, 830, 829, 828, 827, 826, 825, 824, 823, 822, 821, 820, 819, 818, 817, 816, 815, 814, 813, 812, 811, 810, 809, 808, 807, 806, 805, 804, 803, 802, 801, 800, 799
, 798, 797, 796, 795, 794, 793, 792, 791, 790, 789, 788, 787, 786, 785, 784, 783, 782, 781, 780, 779, 778, 777, 776, 775, 774, 773, 772, 771, 770, 769, 768, 767, 766,
765, 764, 763, 762, 761, 760, 759, 758, 757, 756, 755, 754, 753, 752, 751, 750, 749, 748, 747, 746, 745, 744, 743, 742, 741, 740, 739, 738, 737, 736, 735, 734, 733, 73
2, 731, 730, 729, 728, 727, 726, 725, 724, 723, 722, 721, 720, 719, 718, 717, 716, 715, 714, 713, 712, 711, 710, 709, 708, 707, 706, 705, 704, 703, 702, 701, 700, 699,
698, 697, 696, 695, 694, 693, 692, 691, 690, 689, 688, 687, 686, 685, 684, 683, 682, 681, 680, 679, 678, 677, 676, 675, 674, 673, 672, 671, 670, 669, 668, 667, 666, 66
5, 664, 663, 662, 661, 660, 659, 658, 657, 656, 655, 654, 653, 652, 651, 650, 649, 648, 647, 646, 645, 644, 643, 642, 641, 640, 639, 638, 637, 636, 635, 634, 633, 632,
631, 630, 629, 628, 627, 626, 625, 624, 623, 622, 621, 620, 619, 618, 617, 616, 615, 614, 613, 612, 611, 610, 609, 608, 607, 606, 605, 604, 603, 602, 601, 600, 599, 59
8, 597, 596, 595, 594, 593, 592, 591, 590, 589, 588, 587, 586, 585, 584, 583, 582, 581, 580, 579, 578, 577, 576, 575, 574, 573, 572, 571, 570, 569, 568, 567, 566, 565,
564, 563, 562, 561, 560, 559, 558, 557, 556, 555, 554, 553, 552, 551, 550, 549, 548, 547, 546, 545, 544, 543, 542, 541, 540, 539, 538, 537, 536, 535, 534, 533, 532, 53
, 530, 529, 528, 527, 526, 525, 524, 523, 522, 521, 520, 519, 518, 517, 516, 515, 514, 513, 512, 511, 510, 509, 508, 507, 506, 505, 504, 503, 502, 501, 500, 499, 498,
497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480, 479, 478, 477, 476, 475, 474, 473, 472, 471, 470, 469, 468, 467, 466, 465, 46
4, 463, 462, 461, 460, 459, 458, 457, 456, 455, 454, 453, 452, 451, 450, 449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 435, 434, 433, 432, 431,
430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 418, 417, 416, 415, 414, 413, 412, 411, 410, 409, 408, 407, 406, 405, 404, 403, 402, 401, 400, 399, 398, 397
, 396, 395, 394, 393, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382, 381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 370, 369, 368, 367, 366, 365, 364,
363, 362, 361, 360, 359, 358, 357, 356, 355, 354, 353, 352, 351, 350, 349, 348, 347, 346, 345, 344, 343, 342, 341, 340, 339, 338, 337, 336, 335, 334, 333, 332, 331, 33
0, 329, 328, 327, 326, 325, 324, 323, 322, 321, 320, 319, 318, 317, 316, 315, 314, 313, 312, 311, 310, 309, 308, 307, 306, 305, 304, 303, 302, 301, 300, 299, 298, 297,
296, 295, 294, 293, 292, 291, 290, 289, 288, 287, 286, 285, 284, 283, 282, 281, 280, 279, 278, 277, 276, 275, 274, 273, 272, 271, 270, 269, 268, 267, 266, 265, 264, 263
, 262, 261, 260, 259, 258, 257, 256, 255, 254, 253, 252, 251, 250, 249, 248, 247, 246, 245, 244, 243, 242, 241, 240, 239, 238, 237, 236, 235, 234, 233, 232, 231, 230,
229, 228, 227, 226, 225, 224, 223, 222, 221, 220, 219, 218, 217, 216, 215, 21
```

Compiler Practical

```

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
9, 128, 127, 126, 125, 124, 123, 122, 121, 120, 119, 118, 117, 116, 115, 114, 113, 112, 111, 110, 109, 108, 107, 106, 105, 104, 103, 102, 101, 100, 99, 98, 97, 96, 95,
94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53,
52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11,
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
30 : 58 : 575841
1648443658.607041
[999, 999, 997, 996, 995, 994, 993, 992, 991, 990, 989, 988, 987, 986, 985, 984, 983, 982, 981, 980, 979, 978, 977, 976, 975, 974, 973, 972, 971, 970, 969, 968, 967, 9
965, 964, 963, 962, 961, 960, 959, 958, 957, 956, 955, 954, 953, 952, 951, 950, 949, 948, 947, 946, 945, 944, 943, 942, 941, 940, 939, 938, 937, 936, 935, 934, 933
, 932, 931, 930, 929, 928, 927, 926, 925, 924, 923, 922, 921, 920, 919, 918, 917, 916, 915, 914, 913, 912, 911, 910, 909, 908, 907, 906, 905, 904, 903, 902, 901, 900,
899, 898, 897, 896, 895, 894, 893, 892, 891, 890, 889, 888, 887, 886, 885, 884, 883, 882, 881, 880, 879, 878, 877, 876, 875, 874, 873, 872, 871, 870, 869, 868, 867, 86
6, 865, 864, 863, 862, 861, 860, 859, 858, 857, 856, 855, 854, 853, 852, 851, 850, 849, 848, 847, 846, 845, 844, 843, 842, 841, 840, 839, 838, 837, 836, 835, 834, 833,
832, 831, 830, 829, 828, 827, 826, 825, 824, 823, 822, 821, 820, 819, 818, 817, 816, 815, 814, 813, 812, 811, 810, 809, 808, 807, 806, 805, 804, 803, 802, 801, 800, 799
, 798, 797, 796, 795, 794, 793, 792, 791, 790, 789, 788, 787, 786, 785, 784, 783, 782, 781, 780, 779, 778, 777, 776, 775, 774, 773, 772, 771, 770, 769, 768, 767, 766,
765, 764, 763, 762, 761, 760, 759, 758, 757, 756, 755, 754, 753, 752, 751, 750, 749, 748, 747, 746, 745, 744, 743, 742, 741, 740, 739, 738, 737, 736, 735, 734, 733, 73
2, 731, 730, 729, 728, 727, 726, 725, 724, 723, 722, 721, 720, 719, 718, 717, 716, 715, 714, 713, 712, 711, 710, 709, 708, 707, 706, 705, 704, 703, 702, 701, 700, 699,
698, 697, 696, 695, 694, 693, 692, 691, 690, 689, 688, 687, 686, 685, 684, 683, 682, 681, 680, 679, 678, 677, 676, 675, 674, 673, 672, 671, 670, 669, 668, 667, 666, 66
5, 664, 663, 662, 661, 660, 659, 658, 657, 656, 655, 654, 653, 652, 651, 650, 649, 648, 647, 646, 645, 644, 643, 642, 641, 640, 639, 638, 637, 636, 635, 634, 633, 632,
631, 630, 629, 628, 627, 626, 625, 624, 623, 622, 621, 620, 619, 618, 617, 616, 615, 614, 613, 612, 611, 610, 609, 608, 607, 606, 605, 604, 603, 602, 601, 600, 599, 598
, 597, 596, 595, 594, 593, 592, 591, 590, 589, 588, 587, 586, 585, 584, 583, 582, 581, 580, 579, 578, 577, 576, 575, 574, 573, 572, 571, 570, 569, 568, 567, 566, 565,
564, 563, 562, 561, 560, 559, 558, 557, 556, 555, 554, 553, 552, 551, 550, 549, 548, 547, 546, 545, 544, 543, 542, 541, 540, 539, 538, 537, 536, 535, 534, 533, 532, 53
1, 530, 529, 528, 527, 526, 525, 524, 523, 522, 521, 520, 519, 518, 517, 516, 515, 514, 513, 512, 511, 510, 509, 508, 507, 506, 505, 504, 503, 502, 501, 500, 499, 498,
497, 496, 495, 494, 493, 492, 491, 490, 489, 488, 487, 486, 485, 484, 483, 482, 481, 480, 479, 478, 477, 476, 475, 474, 473, 472, 471, 470, 469, 468, 467, 466, 465, 46
4, 463, 462, 461, 460, 459, 458, 457, 456, 455, 454, 453, 452, 451, 450, 449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 435, 434, 433, 432, 431,
430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 418, 417, 416, 415, 414, 413, 412, 411, 410, 409, 408, 407, 406, 405, 404, 403, 402, 401, 400, 399, 398, 397
, 396, 395, 394, 393, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382, 381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 370, 369, 368, 367, 366, 365, 364,
363, 362, 361, 360, 359, 358, 357, 356, 355, 354, 353, 352, 351, 350, 349, 348, 347, 346, 345, 344, 343, 342, 341, 340, 339, 338, 337, 336, 335, 334, 333, 332, 331, 33
0, 329, 328, 327, 326, 325, 324, 323, 322, 321, 320, 319, 318, 317, 316, 315, 314, 313, 312, 311, 310, 309, 308, 307, 306, 305, 304, 303, 302, 301, 30
```