CSE 104L: Programming Lab
Training Set 05 – Python Final Lab Session
Year: 2019 – August-December

LNMIIT, Jaipur

Notes

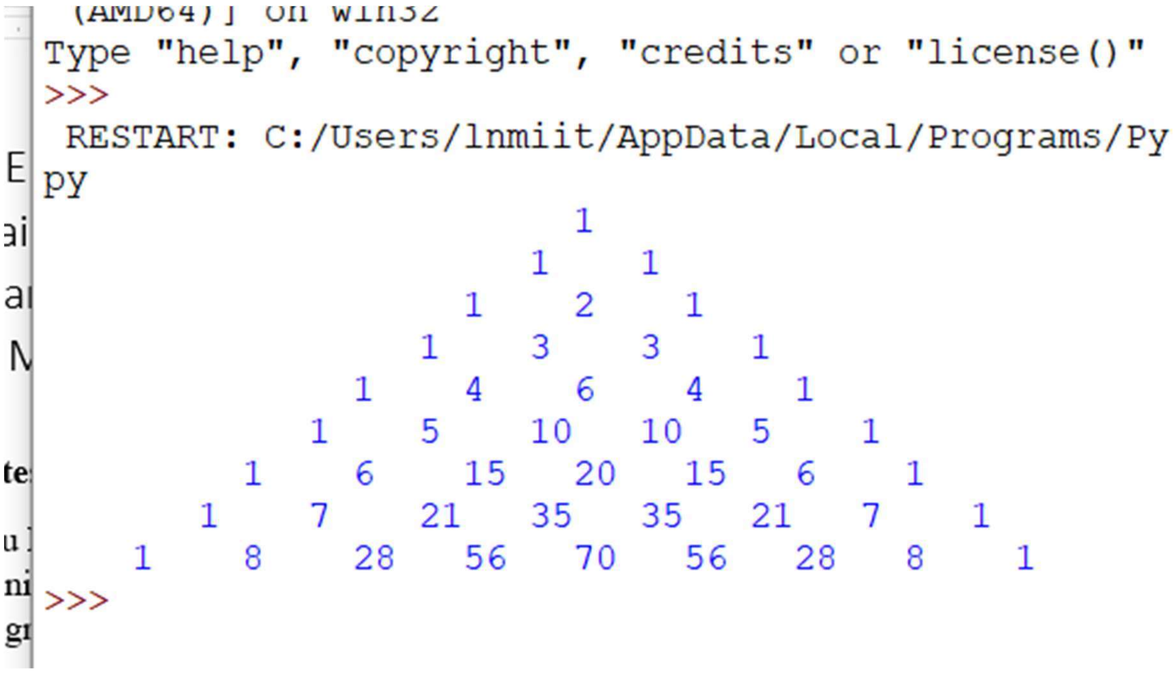
You MUST complete all Tasks listed in Training Sets 01 to 04 before starting on this training. If you have not yet learned the basics you may find this part very difficult and your progress may be unsatisfactory.

CSE104L level 1 Examination will be derived from problems included in Training Sets 01 to 08. Some of the sets are marked as Programming in C. **Students planning to do Python/IDLE examination must practice on these examples (suggested under introductory C training) using IDLE/Python to be best prepared for the lab examination.** Practice using these additional problems will improve students learning and will better prepare the students for Level 1 Examination.

Task 01: Consider Python program with a set of 10 (not necessarily distinct) `int` objects referred by identifiers `a`, `b`, `c`, …, `j`. Complete this Python program without using any Python composite object (for example, list, set or similar) to initialise Python identifiers `A`, `B`, `C`, …, `I` with the objects from the original set. As you note there are only 9 identifiers in the later set of identifiers and we must lose one object from the original set of objects. The object that you should lose is the one with the minimum `int` value.

For example, if the full collection of 10 objects was 0, 1, 0, 2, 3, 1, 2, 6, 7, 8 then new collection will have one less 0.

Task 02: Shown below is a Pascal triangle printed by a Python program.



```
Type "help", "copyright", "credits" or "license()"
>>>
 RESTART: C:/Users/lnmiit/AppData/Local/Programs/Py
py
                        1
                     1     1
                  1     2     1
               1     3     3     1
            1     4     6     4     1
         1     5    10    10     5     1
      1     6    15    20    15     6     1
   1     7    21    35    35    21     7     1
1     8    28    56    70    56    28     8     1
>>>
```

As you note there are 9 rows, and all `int` values are less than 99. Each number is located in a field of 5 character-places.

For printing `int` values, Python function `print()` as default uses one place for sign (for positive integers sign position is left blank) and writes one space after the value as a separator. Thus, a single digit number will use three places, two digit numbers will use 4 places.

You also note that numbers in alternate rows are aligned on two separate column sets.

Python function `print()` discussed in the class has a default behaviour that adds next-line character after values of its arguments are printed. We can stop this behaviour by adding an argument `end` to the call as follows:

```
print(values-to-be-printed, end='')
```

Here is an example execution on IDLE platform.

Python script

```
    x = 10
    y = 15
    print('if x = ', x, 'and ',end='')
    print('y = ', y, 'then ',end='')
    print('x + y = ', x+y)
```

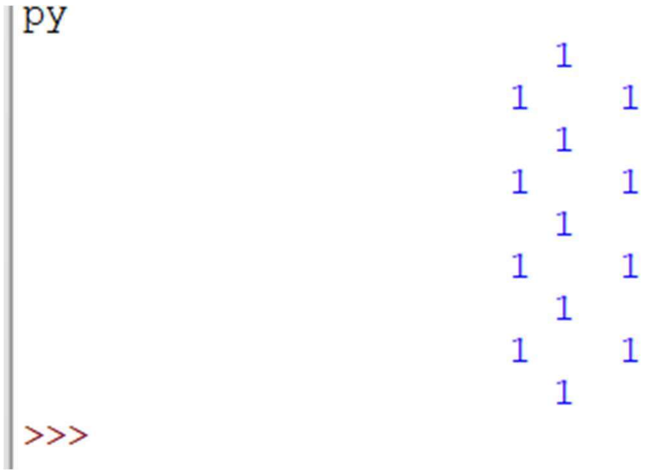
Output:

```
=== RESTART:
C:/Users/lnmiit/AppData/Local/Programs/Python/Python38/xy.py ===
if x =  10 and y =  15 then x + y =  25
```

You are given a list `row` defined as below.

```
      row = [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0]
```

You will be required to write Python commands to print the row 9 times as shown in the picture:



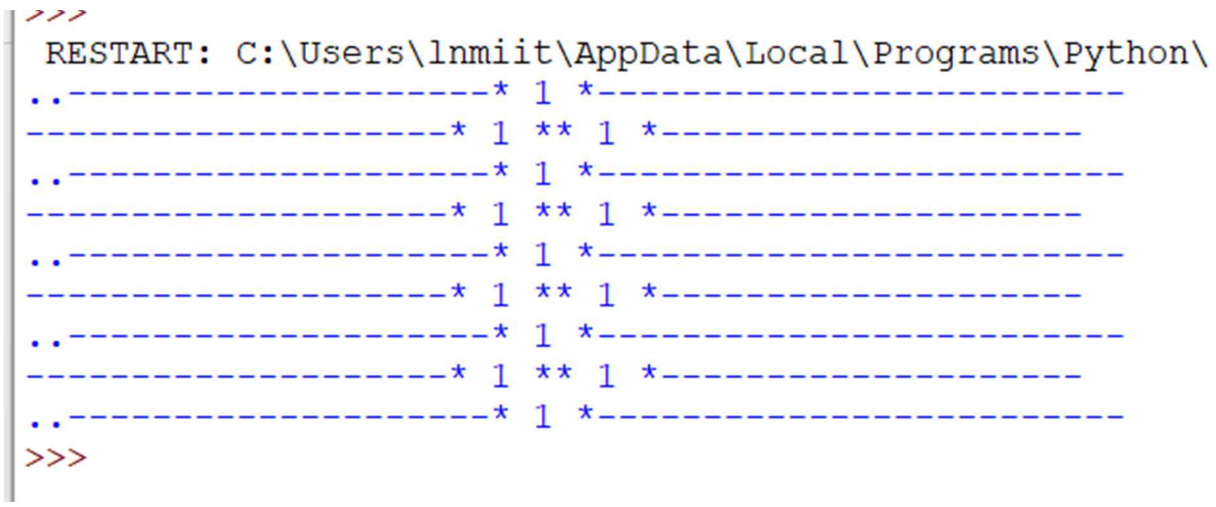
```
py
                        1
                      1    1
                        1
                      1    1
                        1
                      1    1
                        1
                      1    1
                        1
>>>
```

The list is printed 9 times using an outer loop. Each odd iteration of the loop prints list elements from the odd index locations. Even numbered iterations print elements at the even indexed locations. This is done by loop(s) nested in the outer loop. The inner loop prints relevant elements from the list `row`.

Value 0 is printed as 5 spaces. Single digit integers will use 3 positions (explained previously), we add a space before and a space after these three positions controlled by function `print()`. Additionally, each even row is shifted by two spaces to get the serrated pattern visible in the picture above.

To help you in writing your code to print the above pattern, you are provided another output where programmer replaced spaces by strings of characters `.`, `*`, and `-`.

Dots (.) show the spaces that were added to offset even numbered row by two positions. Stars (*) show the positions where additional spaces were added around a single digit `int` value. Dashes (-) are in groups of five and are inserted to mark the 5 spaces printed for value 0. There is no example here for case where printed value had two digits.



```
>>>
 RESTART: C:\Users\lnmiit\AppData\Local\Programs\Python\
..--------------------* 1 *-------------------------
--------------------* 1 ** 1 *--------------------
..--------------------* 1 *-------------------------
--------------------* 1 ** 1 *--------------------
..--------------------* 1 *-------------------------
--------------------* 1 ** 1 *--------------------
..--------------------* 1 *-------------------------
--------------------* 1 ** 1 *--------------------
..--------------------* 1 *-------------------------
>>>
```

Write Python code to print the row 9 time in the form shown above. You will need to check the number of digits in the value to be printed to determine how you include spaces in around the printed values.

Task 03: Once you can get the right printout for Task 02, try it for different values of row list. Some suggested test lists are (three separate test cases):

```
row = [0,0,0,0,0,10,0,0,0,1,1,1,0,0,0,0,10,0,0,0,0,0]
row = [0,0,0,0,0,0,0,0,0,26,1,26,0,0,0,0,0,0,0,0,0,0]
row = [0,0,0,12,0,0,0,0,0,1,1,1,0,0,0,0,0,0,12,0,0,0]
```

Task 04: Printing full Pascal Triangle is now very easy. In each iteration of the loop, the elements from list `row` not being printed are updated as follows.

```
row[i] = row[i-1]+row[i+1]
```

This is the last Training Set for Python part of the subject.