TRAINING DAY 02 REPORT

24 June, 2025

Topic : Introduction to Python Programming – Basics

Objectives of the Day
•Introduction to Functions in Python
• Lists and Tuples
• Basic programs using lists
• Problem-solving with functions

Lists in Python

Lists are ordered, mutable collections of items in Python. They are versatile and can store items of different data types.

Key characteristics of lists:

- Ordered: Items maintain their insertion order.
- Mutable: Elements can be changed, added, or removed after creation.
- Indexed: Elements are accessed using numerical indices, starting from 0 for the first element. Negative indexing can be used to access elements from the end of the list.
- Heterogeneous: Lists can contain items of different data types (e.g., integers, strings, other lists).

Basic list operations:

- Creation: `my_list = [1, "hello", 3.14]`
- Accessing elements: `my_list[0]` (first element), `my_list[-1]` (last element)
- Modifying elements: `my_list[0] = 10`
- Adding elements: `my_list.append(4)`, `my_list.insert(1, "world")`
- Removing elements: `my_list.remove("hello")`, `my_list.pop(0)`
- Slicing: `my_list[1:3]` (elements from index 1 up to, but not including, index 3)

Pattern Printing in Python

Pattern printing involves generating various visual patterns using characters (like `*`, `#`, numbers, or letters) arranged in specific shapes. This is typically achieved using loops, especially nested loops.

Theoretical basis of pattern printing:

- **Nested Loops:**

  The core of most pattern printing programs relies on nested loops.

- Outer Loop: Controls the number of rows in the pattern.
- Inner Loop(s): Control the elements printed within each row, often handling spaces and the pattern characters themselves.
- **Controlling Output:**
- `print()`: Used to display characters or numbers.
- `end=""`: The `end` parameter in `print()` prevents a new line after printing, allowing elements to be printed on the same line.
- `print()` without arguments: Used after the inner loop to move to the next line for the next row of the pattern.
- **Logic for specific patterns:**
- Triangles/Pyramids: The number of characters or spaces in the inner loop often depends on the current row number from the outer loop.
- Hollow Patterns: Conditional statements (e.g., `if-else`) are used within the inner loop to determine whether to print a character or a space based on the position within the pattern's boundaries.
- Number/Alphabet Patterns: `chr()` function can be used to convert ASCII values to characters for alphabet patterns. Mathematical calculations are employed to determine the numbers to be printed in number patterns.

programs covered :

```
'''
1. write a program to print following pattern
   1
  1 2
 1 2 3
1 2 3 4
'''
from ctypes.wintypes import tagMSG

s=" "
n=4
a=""
for i in range(1,5):
    x=n-i
    a = a + str(i) + " "
    print(x *s ,a, ' ')
print()
```

```
   1
  1 2
 1 2 3
1 2 3 4
```

```
'''
2. write a program to print following pattern
10 9 8 7
6 5 4
3 2
1
'''
c=11
for i in range(4,0,-1):
    for j in range(i):
        c=c-1
        print(c,end=" ")
    print()
print()
```

```
10 9 8 7
6 5 4
3 2
1
```

```
'''
3. write a program to print following pattern
1
2 1
1 2 3
4 3 2 1
'''

def reverse(num):
    sum = 0
    while (num != 0):
        r = int (num % 10)
        sum = sum * 10 + r
        num = int(num / 10)
    return str(sum)

s = ""
for i in range(1,5):
    if(i%2==0):
        s = s + str(i)
        k = reverse(int(s))
        print(k)
    else:
        s=s + str(i)
        print(s)
print()
```

```
1
21
123
4321
```

```
'''
4. write a program to print following pattern
10 9 8 7
4 5 6
3 2
1
'''
x = 10
for i in range(4,0,-1):
    tmp = ""
    s=" "
    for j in range(i):

        if i % 2 == 0:
            tmp = tmp + str(x) + s
        else:
            tmp =   str(x)+ s + tmp
        x = x - 1
    print(tmp)
print()
```

```
10 9 8 7
4 5 6
3 2
1
```

Summary of Day 02

The second day of training focused on building a strong foundation in Python programming by covering essential concepts like **functions**, **lists**, **tuples**, and **pattern printing**. Participants learned how to create and manipulate lists using various built-in methods and practiced writing basic programs involving list operations. They also explored how to use **functions** effectively for structuring and reusing code.

The session further introduced **pattern printing**, emphasizing logic building through the use of **nested loops**, **print formatting**, and **conditional statements**. Participants engaged in practical implementation of different types of patterns including pyramids, number sequences, and hollow shapes.