Introduction

Patterns are a handy application of loops and will provide you with better clarity and understanding of the implementation of loops.

Before printing any pattern, you must consider the following three things:

- The first step in printing any pattern is to figure out the number of rows that the pattern requires.
- Next, you should know how many columns are there in the i^{th} row.
- Once, you have figured out the number of rows and columns, then focus on the pattern to print.

For eg. We want to print the following pattern for N rows: **(Pattern 1.1)**

```
#For N=4:
****
****
****
****
```

Approach:

From the above pattern, we can observe:

➔ **Number of Rows:** The pattern has 4 rows. We have to print the pattern for N rows.

➔ **Number of Columns:** All the rows have 4 columns. Thus, in a pattern of N rows, all the rows will have N columns.

➔ **What to print:** We have to print `*` 4 times in all the 4 rows. Thus, in a pattern of N rows, we will have to print `*` N times in all the rows.

Now, let us discuss how to implement such patterns using Python.

Python Implementation for Patterns

We generally need two loops to print patterns. The outer loop iterates over the rows, while the inner nested loop is responsible for traversing the columns. The **algorithm** to print any pattern can be described as follows:

- Accept the number of rows or size of the pattern from a user using the `input()` function.
- Iterate the rows using the outer loop.
- Use the nested inner loop to handle the column contents. The internal loop iteration depends on the values of the outer loop.
- Print the required pattern contents using the `print()` function.
- Add a new line after each row.

The implementation of **Pattern 1.1** in Python will be:

Step 1: Let us first use a loop to traverse the rows. This loop will start at the first row and go on till the N^{th} row. Below is the implementation of this loop:

```
N=int(input()) #Take user input, N= Number of Rows
row=1; #The loop starts with the 1st row
while row<=N: #Loop will on for N row
    #<Here goes the Nested Loop>
    row=row+1 #Increment the current row (Outer Loop)
    print() #Add a new Line after each row
```

Printing a New Line: Since we need to print the pattern in multiple lines, we will have to add a new line after each row. Thus for this purpose, we use an empty `print()` statement. The `print()` function in Python, by default, ends in a new line.

Step 2: Now, we need another loop to traverse the row during each iteration and print the pattern; this can be done as follows:

```
N=int(input()) #Take user input, N= Number of Rows
row=1; #The loop starts with the 1st row
while row<=N: #Loop will on for N rows
     col=1; #The loop starts with the first column in the current row
    while col<=N: #Loop will on for N columns
        print("*",end="") #Printing a (*) in all columns
        col=col+1 #Increment the current column (Inner Loop)
    row=row+1 #Increment the current row (Outer Loop)
    print() #Add a new Line after each row is printed
```

Printing in the same line:

The `print()` function in Python, by default, ends in a new line. This means that `print("*")`, would print `*` and a new line character. Now if anything is printed after this, it will be printed in a new line. However, If we have to print something in the same line, we will have to pass another argument (`end=`) in the `print()` statement. Thus, when we write the command `print("*", end="")`, Python prints a `*` and it ends in an empty string instead of a new line; this means that, when the next thing is printed, it will be printed in the same line as `*`.

There are two popular types of patterns-related questions that are usually posed:

- Square Pattern - **Pattern 1.1** is square.
- Triangular Pattern