Patterns

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 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

Let us now look at the implementation of some common patterns.

Square Patterns

Pattern 1.2

```
# N = 5
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Approach:
From the above pattern **we can observe:**

➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.

➔ **Number of Columns:** All the rows have 5 columns. Thus, in a pattern of N rows, all the rows will have N columns.

➔ **What to print:** All the entries in any row, are the same as the corresponding row numbers. Thus in a pattern of N rows, all the entries of the i^{th} row are i (1^{st} row has all 1's, 2^{nd} row has all 2's, and so on).

Python Implementation:

```
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(row,end="") #Printing the row number 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
```

Pattern 1.3

```
# N = 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Approach:

From the above pattern, **we can observe:**

- ➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** All the rows have 5 columns. Thus, in a pattern of N rows, all the rows will have N columns.
- ➔ **What to print:** All the entries in any row, are the same as the corresponding column numbers. Thus in a pattern of N rows, all the entries of the i^{th} column are i (1^{st} column has all 1's, 2^{nd} column has all 2's, and so on).

Python Implementation:

```
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(col,end="") #Printing the column number 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
```

Pattern 1.4

```
# N = 5
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
```

Approach:

From the above pattern, **we can observe:**

- ➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** All the rows have 5 columns. Thus, in a pattern of N rows, all the rows will have N columns.
- ➔ **What to print:** All the entries in any row, are N-columnNumber+1. Thus in a pattern of N rows, all the entries of the i^{th} column are N-i+1 (1^{st} column has all 5's (5-1+1), 2^{nd} column has all 4's (5-2+1), and so on).

Python Implementation:

```
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(N-col+1,end="") #Printing (N-col+1) 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
```

This way there can be several other square patterns and you can easily print them using this approach- **By finding the number of Rows, Columns, and What to print**.

Pattern 1.5

```
# N = 5
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

Approach:

From the above pattern, **we can observe:**

- ➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** All the rows have 5 columns. Thus, in a pattern of N rows, all the rows will have N columns.
- ➔ **What to print:** The first entry in the 1^{st} row is 1, the first entry in the 2^{nd} row is 2, and so on. Further, these values are incremented continuously by 1 in the remaining entries of any particular row. Thus in a pattern of N rows, the first entry of the i^{th} row is i. The remaining entries in the i^{th} row are i+1,i+2, **and** so on. It can be observed that any entry in this pattern can be written as row+col-1.

```
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(col+row-1,end="") #Printing row+col-1 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
```

Triangular Patterns

Pattern 1.6

```
# N = 5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Approach:

From the above pattern **we can observe:**

- ➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** The number of columns in any row is the same as the corresponding row number.1st row has 1 column, 2nd row has 2 columns, and so on. Thus, in a pattern of N rows, the ith row will have i columns.
- ➔ **What to print:** All the entries in any row, are the same as the corresponding row numbers. Thus in a pattern of N rows, all the entries of the ith row are i (1st row has all 1's, 2nd row has all 2's, and so on).

Python Implementation:

```
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<=row: # Number of cols = The row number
        print(row,end="") #Printing the row number 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
```

Pattern 1.7

```
# N = 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Approach:

From the above pattern **we can observe:**

- ➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** The number of columns in any row is the same as the corresponding row number.1st row has 1 column, 2nd row has 2

columns, and so on. Thus, in a pattern of N rows, the i^{th} row will have i columns.

➔ **What to print:** All the entries in any row, are the same as the corresponding column numbers. Thus in a pattern of N rows, all the entries of the i^{th} column are i (1^{st} column has all 1's, 2^{nd} column has all 2's, and so on).

Python Implementation:

```
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<=row: # Number of cols = The row number
        print(col,end="") #Printing the column number 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
```

Pattern 1.8

```
# N = 5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Approach:

From the above pattern, **we can observe:**

➔ **Number of Rows:** The pattern has 5 rows. We have to print the pattern for N rows.

➔ **Number of Columns:** The number of columns in any row is the same as the corresponding row number.1^{st} row has 1 column, 2^{nd} row has 2 columns, and so on. Thus, in a pattern of N rows, the i^{th} row will have i columns.

➔ **What to print:** The pattern starts with 1 and then each column entry is incremented by 1. Thus, we will initialize a variable temp=1. We will keep printing the value of temp in the successive columns and upon printing, we will increment the value of temp by 1.

Python Implementation:

```
N=int(input()) #Take user input, N= Number of Rows
row=1; #The loop starts with the 1st row
temp=1
while row<=N: #Loop will on for N rows
    col=1; #The loop starts with the first column in the current row
    while col<=row: #Number of cols = The row number
        print(temp,end="") #Printing value of temp in all columns
        temp=temp+1
        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
```

Character Patterns

Pattern 1.9

```
# N = 4
ABCD
ABCD
ABCD
ABCD
```

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:** The 1st column has all A's, 2nd column has all B's, and so on. The **ASCII** value of A is 65. In the 1st column, the character corresponds to the **ASCII** value 65 (64+1). In the 2nd column, the character corresponds to the **ASCII** value 66 (64+2). Thus, all the entries in the i^{th} column are equal to the character corresponding to the **ASCII** value **64+i**. The chr() function gives the character associated with the integral ASCII value within the parentheses.

Python Implementation:

```
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(chr(64+col),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
```

Pattern 1.10

```
# N = 4
ABCD
BCDE
CDEF
DEFG
```

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:** This pattern is very similar to **Pattern 1.5.** We can implement this using a similar code with a minor change. Instead of integers, we need capital letters of the same order. Instead of 1, we need A, instead of 2, we need B and so on. **ASCII** value of A is 65. Thus if we add 64 to all the entries in **Pattern 1.5** and find their **ASCII** values, we will get our result. The chr() function gives the character associated with the integral **ASCII** value within the parentheses.

Python Implementation:

```
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(chr(64+col+row-1),end="") # Adding 64 to all entries
        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
```

Some Advanced Patterns

Pattern 2.1 - Inverted Triangle

```
# N = 3
* * *
* *
*
```

Approach:

From the above pattern, **we can observe:**

- ➔ **Number of Rows:** The pattern has 3 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** The number of columns in any row is equal to **N-rowNumber+1**.1st row has 3 columns (3-1+1), 2nd row has 2 columns (3-2+1), and so on. Thus, in a pattern of N rows, the ith row will have N-i+1 columns.
- ➔ **What to print:** All the entries in any row are "*".

Python Implementation:

```
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-row+1: #Number of columns = N-rowNumber+1
        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
```

Pattern 2.2 - Reversed Pattern

```
# N = 3
    *
  * *
* * *
```

Approach:
From the above pattern, **we can observe:**

- ➔ **Number of Rows:** The pattern has 3 rows. We have to print the pattern for N rows.
- ➔ **Number of Columns:** The number of columns in any row is equal to **N**.
- ➔ **What to print:** In the 1st row, while **columnNumber <= 2(3-1)**, we print a **" "** in every column. Beyond the 2nd column, we print a **"*"**. Similarly, in the 2nd row, we print a **" "** till **columnNumber <=1(3-2)** and beyond the 1st column, we print a **"*"**. We can easily notice that if **col <= N-rowNumber**, we are printing a **" " (Space)**. And if **col > N-rowNumber**, we are printing a **"*"**.

Python Implementation:

```
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:#The loop will go on for N columns
        if(col<=N-row):
            print(" ",end="") #Printing a (" ")
        else:
            print("*",end="") #Printing a (*)
        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
```

Pattern 2.3 - Isosceles Pattern

```
# N = 4
   1
  121
 12321
1234321
```

Approach:
From the above pattern **we can observe:**

- ➔ **Number of Rows:** The pattern has 3 rows. We have to print the pattern for N rows.

- ➔ **Number of Columns:** Similar to Pattern 2.2, we first have **N-rowNumber** columns of spaces. Following this, we have **2*rowNumber-1** columns of numbers.
- ➔ **What to print:** We can notice that if **col <= N-rowNumber**, we are printing a **" " (Space)**. Further, the pattern has two parts. First is the increasing part and second is the decreasing part. For the increasing part, we will initialize a variable **num=1**. In each row, we will keep printing **num** till its value becomes equal to the **row number**. We will increment **num** by 1 after printing it; this will account for the first part of the pattern. We have **num = rowNumber** at this stage. Now, for the decreasing part, we will again start printing **num** till **num>=1**. After printing **num,** we will decrease it by 1.

Python Implementation:

```
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
    spaces =1 # Printing spaces
    while spaces<= N- row:
        print(" ",end="")
        spaces=spaces+1

    num=1 #Variable to print the numbers
    while num<=row: #Increasing Pattern
        print(num,end="")
        num=num+1;

    num=row-1 # We have to start printing the decreasing part from  one less than the rowNumber
    while num>=1: #Decreasing Pattern
        print(num,end="")
        num=num-1
    print()#New Line
    row=row+1
```