Python Loop Fundamentals with Range for Beginners

■ Introduction to Loops in Python

Loops allow us to execute a block of code repeatedly. Python provides two main loop types: 1. **for loop** – used for iterating over a sequence (like a list, tuple, or range) 2. **while loop** – runs as long as a condition remains True

1■■ Python For Loop

A **for loop** is used when you want to iterate through a sequence of items. It is commonly used with the **range()** function to generate a sequence of numbers.

Syntax:

```
for variable in sequence: # code to execute
```

■ The range() Function

The **range()** function generates a sequence of numbers. It is commonly used with for loops to control the number of iterations.

Syntax:

```
range(start, stop, step)
```

- **start**: The beginning value (default is 0)
- **stop**: The end value (not included)
- **step**: The increment value (default is 1)

Example 1:

```
for i in range(5): print(i) Output: 0 1 2 3 4
```

Example 2 (Custom Start and Step):

```
for i in range(2, 10, 2): print(i) Output: 2 4 6 8
```

2■■ Nested For Loop

A **nested loop** is a loop inside another loop. It is useful for handling multi-dimensional data or creating patterns.

Example:

```
for i in range(3): for j in range(2): print(f"i = {i}, j = {j}") Output: i = 0, j = 0
i = 0, j = 1 i = 1, j = 0 i = 1, j = 1 i = 2, j = 0 i = 2, j = 1
```

3■■ Python While Loop

A **while loop** executes a block of code as long as the given condition is True. It's useful when the number of iterations is not predetermined.

Example:

```
count = 0 while count < 5: print("Count:", count) count += 1 Output: Count: 0 Count:
1 Count: 2 Count: 3 Count: 4
```

4■■ Loop Control Statements

Python provides three keywords to control loop flow:

- **break** – Exits the loop completely.

- **continue** – Skips the current iteration and continues with the next one.
- **pass** – Does nothing, used as a placeholder.

Example:

```
for i in range(10): if i == 5: break if i % 2 == 0: continue print("Odd number:", i)
Output: Odd number: 1 Odd number: 3
```

5■■ Else Block with Loops

You can use an **else** statement with loops. The else block executes only if the loop completes normally (without encountering a break statement).

Example:

```
for i in range(3): print("Number:", i) else: print("Loop completed!") Output:
Number: 0 Number: 1 Number: 2 Loop completed!
```

■ Practice Programs

1 1. Print numbers from 1 to 10 using range().

2 2. Print even numbers between 1 and 20 using range().

3 3. Print the multiplication table of 7 using range().

4 4. Print a pattern of stars using nested for loops.

5 5. Find the sum of numbers from 1 to 100 using for loop.

6 6. Print numbers in reverse order using range().

7 7. Display all odd numbers from 1 to 50 using range().

8 8. Calculate the factorial of a number using while loop.

9 9. Use range() with step = -1 to print reverse counting.

10 10. Iterate over a list using for loop and range().

■ Conclusion

The **range()** function and loops are foundational tools in Python programming. They allow you to automate repetitive tasks, iterate efficiently, and create powerful logic flows. Mastering these will make you a confident Python programmer!