Python Loop Fundamentals for Beginners

■ Introduction to Loops in Python

Loops are used in Python to execute a block of code repeatedly. They help in performing repetitive tasks efficiently. There are two main types of loops in Python: 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 to iterate over a sequence (list, tuple, dictionary, string, or range). It runs the block of code once for each element in the sequence.

Syntax:

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

Example:

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

Looping through a List:

```
fruits = ["apple", "banana", "cherry"] for fruit in fruits: print(fruit) Output:
apple banana cherry
```

2■■ Nested For Loop

A **nested loop** means having a loop inside another loop. It is often used to work with multi-dimensional data structures like matrices.

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. Use it when the number of iterations is not known in advance.

Syntax:

```
while condition: # code to execute
```

Example:

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

■■ Be Careful:

If the condition in a while loop never becomes False, it can lead to an infinite loop. Always make sure to update variables within the loop to eventually stop execution.

4■■ Loop Control Statements

Python provides special keywords to control loop execution:

- **break** – Exits the loop completely when a condition is met.
- **continue** – Skips the current iteration and moves to the next one.
- **pass** – Does nothing, used as a placeholder when a statement is required syntactically.

Example using break and continue:

```
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 in Loops

Python loops can have an **else** block which executes only if the loop completes without a **break** statement.

Example:

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

■ Practice Programs

1. 1. Print numbers from 1 to 20 using for loop.
2. 2. Display even numbers between 1 and 50 using while loop.
3. 3. Print a multiplication table using nested loops.
4. 4. Print a triangle pattern using nested for loops.
5. 5. Calculate factorial of a number using while loop.
6. 6. Find the sum of digits of a number using while loop.
7. 7. Display all vowels from a string using for loop.
8. 8. Print Fibonacci series using while loop.
9. 9. Iterate through a dictionary using for loop.
10. 10. Create a loop that breaks when user enters 'exit'.

■ Conclusion

Loops are one of the most essential concepts in Python. Mastering them helps you automate repetitive tasks, handle data efficiently, and build logical flow in your programs. Practice is the key to mastering loops!