Chapter 7 – Loops in Python

Loops are used to execute a block of code repeatedly based on a condition or sequence. In Python, there are two primary types of loops:

1. **While Loops**
2. **For Loops**

While Loop

The `while` loop runs as long as a specified condition is true. Here's the basic syntax:

```
while (condition):
    # Body of the loop
```

Example: Print numbers from 1 to 5

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
```

Note: If the condition never becomes false, the loop will run indefinitely.

Example: Print the contents of a list using a `while` loop

```
my_list = [10, 20, 30]
i = 0
while i < len(my_list):
    print(my_list[i])
    i += 1
```

For Loop

The `for` loop is used to iterate over a sequence (like a list, tuple, or string) or other iterable objects. Here's the basic syntax:

```
for item in sequence:
    # Body of the loop
```

Example: Print each item in a list

```
l = [1, 7, 8]
for item in l:
    print(item)
```

Range Function

The `range()` function generates a sequence of numbers. It can be used with one, two, or three arguments:

```
range(stop)           # From 0 to stop-1
range(start, stop)    # From start to stop-1
range(start, stop, step) # From start to stop-1, incrementing by step
```

Example: Print numbers from 0 to 6

```
for i in range(7):
    print(i)
```

For Loop with Else

An `else` clause can be used with a `for` loop to specify code that should be executed when the loop finishes iterating over all items.

Example:

```
l = [1, 7, 8]
for item in l:
    print(item)
else:
    print("done")
```

Output:

```
1
7
8
done
```

The `break` Statement

The `break` statement exits the loop immediately when encountered.

Example:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Output:

```
0
1
2
3
4
```

The `continue` Statement

The `continue` statement skips the rest of the code inside the loop for the current iteration and moves to the next iteration.

Example:

```
for i in range(4):
    if i == 2:
        continue
    print(i)
```

Output:

```
0
1
3
```

The pass Statement

The `pass` statement is a null operation; it does nothing when executed. It's used as a placeholder for future code.

Example:

```
l = [1, 7, 8]
for item in l:
    pass  # This will not do anything
```

Understanding Nested Loops in Python

Nested loops in Python are loops inside other loops. This concept is used when you need to perform repetitive tasks in multiple dimensions. For example, iterating over a 2D grid, processing multi-dimensional data, or creating complex patterns like the triangle patterns we discussed earlier.

Basic Concept of Nested Loops

- An outer loop runs first.
- For each iteration of the outer loop, the inner loop runs completely.

Syntax:

```
for outer in outer_sequence:
    # Outer loop code block
    for inner in inner_sequence:
        # Inner loop code block
```

Example 1: Multiplication Table

Let's print a multiplication table using nested loops.

```python
for i in range(1, 6):  # Outer loop for rows
    for j in range(1, 6):  # Inner loop for columns
        print(i * j, end='\t')
    print()  # Newline after each row
```

Explanation:

1. **Outer Loop (`for i in range(1, 6)`):**
    - This loop runs 5 times, for `i` values from 1 to 5.
2. **Inner Loop (`for j in range(1, 6)`):**
    - For each `i`, this loop also runs 5 times, for `j` values from 1 to 5.
3. **Print Statement (`print(i * j, end='\t')`):**
    - Multiplies `i` and `j` and prints the result in the same line separated by a tab.
4. **Newline (`print()`):**
    - After the inner loop completes, a newline is printed to start a new row.

Output:

```
1   2   3   4   5
2   4   6   8   10
3   6   9   12  15
4   8   12  16  20
5   10  15  20  25
```

Example 2: Printing a 2D List (Matrix)

Nested loops can be used to print elements of a 2D list (matrix).

```python
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for row in matrix:  # Outer loop for each row
    for element in row:  # Inner loop for each element in the row
        print(element, end=' ')
    print()  # Newline after each row
```

Explanation:

1. **Outer Loop (`for row in matrix`):**
    - This loop iterates through each row in the matrix.
2. **Inner Loop (`for element in row`):**
    - For each row, this loop iterates through each element in the row.
3. **Print Statement (`print(element, end=' ')`):**
    - Prints each element followed by a space, staying on the same line.
4. **Newline (`print()`):**
    - After all elements in a row are printed, a newline is printed to start a new row.

Output:

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

Key Points to Remember

1. **Order of Execution:**
    - The outer loop starts first and runs once for each iteration of the inner loop.
    - The inner loop completes all its iterations for each iteration of the outer loop.
2. **Usage:**
    - Nested loops are useful for working with multi-dimensional data structures (like matrices).
    - They help in tasks where each element needs to be processed within a structured hierarchy.

3. **Complexity:**
    - Be mindful of the time complexity, as nested loops can lead to performance issues with large data sets (O(n^2) complexity).

Nested loops can be powerful tools for solving problems that involve multiple layers of iteration. By understanding the flow of nested loops, students can effectively utilize them to tackle complex programming challenges.