Lab 12: PYTHON LOOP

Lab Objectives

By the end of this lab, students will be able to:

- Understand and use for and while loops effectively.
- Apply loop control statements (break, continue, pass).
- Use loops to solve computational problems.

Loop In Python:

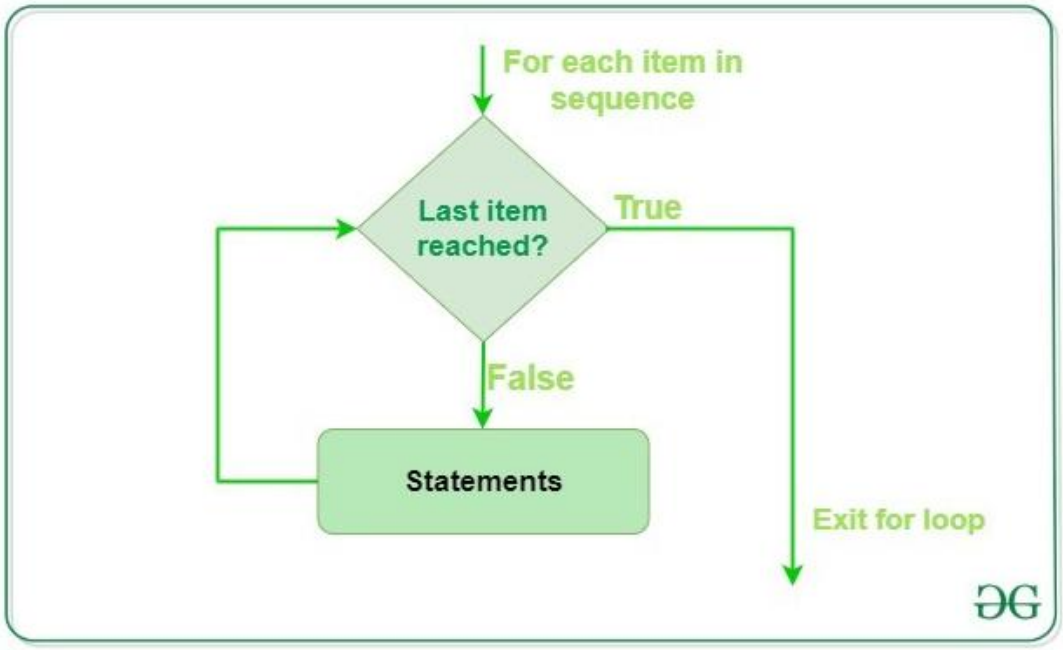
Loops in Python are used to repeat actions efficiently. The main types are For loops (counting through items) and While loops (based on conditions).

Loops are used to execute a block of code repeatedly until a certain condition is met. They help reduce redundancy, make programs more efficient, and allow automation of repetitive tasks. In continue, and pass.

Concept	Description	Syntax
for loop	Executes a block of code for each value in a sequence.	for i in range(start, stop, step):
while loop	Repeats while a condition is true.	while condition:
break	Exits the loop early.	Break
continue	Skips to the next iteration.	continue pass
pass	Placeholder, does nothing.	Pass

For Loop:

For loops is used to iterate over a sequence such as a list, tuple, string or range. It allow to execute a block of code repeatedly, once for each item in the sequence



EXAMPLES:

main.py

Share Run

```
1  n = 4
2  for i in range(0, n):
3      print(i)
4
5
6
7
8
```

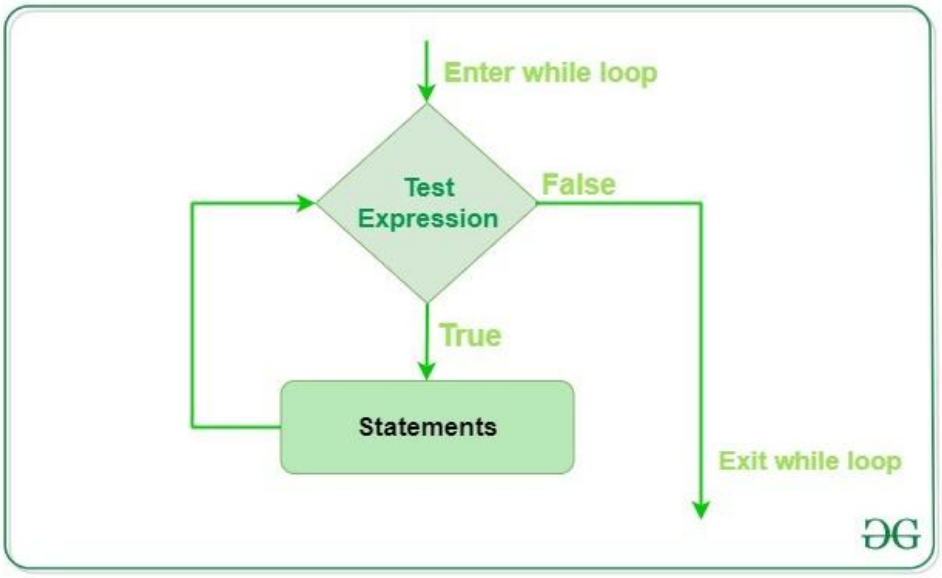
Output Clear

```
0
1
2
3

=== Code Execution Successful ===
```

While Loop:

In Python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.



main.py

Share Run

```
1  # Print numbers from 1 to 5 using while loop
2  i = 1
3  while i <= 5:
4      print(i)
5      i += 1
6
```

Output Clear

```
1
2
3
4
5

=== Code Execution Successful ===
```

Example:

Iterating Over List, Tuple, String and Dictionary Using for Loops in Python

```
li = ["red", "green", "yellow"]
for x in li:
    print(x)

tup = ("red", "green", "yellow")
for x in tup:
    print(x)

s = "abc"
for x in s:
    print(x)

d = dict({'x':123, 'y':354})
for x in d:
    print("%s  %d" % (x, d[x]))

set1 = {10, 30, 20}
for x in set1:
    print(x),
```

Output

```
red
green
yellow
red
green
yellow
a
b
c
x  123
y  354
10
20
30
```

Iterating by Index of Sequences

We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and then iterate over the sequence within the range of this length.

```
li = ["red", "green", "yello"]
for index in range(len(li)):
    print(li[index])
```

Output

```
red
green
yellow
```

Explanation: This code iterates through each element of the list using its index and prints each element one by one. The **range(len(list))** generates indices from 0 to the length of the list minus 1.

Infinite While Loop

If we want a block of code to execute infinite number of times then we can use the while loop in Python to do so.

Code given below uses a 'while' loop with the condition "**True**", which means that the loop will run infinitely until we break out of it using "**break**" keyword or some other logic.

```
while (True):
    print("Hello World!")
```

Note: *It is suggested not to use this type of loop as it is a never-ending infinite loop where the condition is always true and we have to forcefully terminate the compiler.*

Nested Loops

Python programming language allows to use one loop inside another loop which is called nested loop. Following example illustrates the concept.

```
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

Output

```
1

2 2

3 3 3

4 4 4 4
```

Explanation: In the above code we use nested loops to print the value of i multiple times in each row, where the number of times it prints i increases with each iteration of the outer loop. The print() function prints the value of i and moves to the next line after each row.

A final note on loop nesting is that we can put any type of loop inside of any other type of loops in Python. For example, a for loop can be inside a while loop or vice versa.

Loop Control Statements

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement

The continue statement in Python returns the control to the beginning of the loop.

```
for letter in 'geeksforgeeks':
   if letter == 'e' or letter == 's':
      continue
   print('Current Letter :', letter)
```

Output

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Explanation: The continue statement is used to skip the current iteration of a loop and move to the next iteration. It is useful when we want to bypass certain conditions without terminating the loop.

Break Statement

The break statement in Python brings control out of the loop.

```
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
        break

print('Current Letter :', letter)
```

Output

```
Current Letter : e
```

Explanation: break statement is used to exit the loop prematurely when a specified condition is met. In this example, the loop breaks when the letter is either 'e' or 's', stopping further iteration.

Pass Statement

We use pass statement in Python to write empty loops. Pass is also used for empty control statements, functions and classes.

```
for letter in 'hello world':
    pass
print('Last Letter :', letter)
```

Output

```
Last Letter : d
```

Explanation: In this example, the loop iterates over each letter in 'geeksforgeeks' but doesn't perform any operation, and after the loop finishes, the last letter ('s') is printed.

Example 1:

```
# Calculate the sum of first 5 natural numbers
total = 0
for i in range(1, 6):
    total += i
print("Sum =", total)
```

Output

```
Sum = 15
```

Example 2: (Using if-else with Loop – Even and Odd Numbers)

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i, "is Even")
    else:
        print(i, "is Odd")
```

Output

```
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
6 is Even
7 is Odd
8 is Even
9 is Odd
10 is Even
```

EXERCISE:

1. Write a Python code to print number patter using a loop.

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2. Write a Python program to display only those numbers from a list that satisfy the following conditions

1. The number must be divisible by five
2. If the number is greater than 150, then skip it and move to the following number
3. If the number is greater than 500, then stop the loop

Examples:

Input: numbers = [12, 75, 150, 180, 145, 525, 50]

Output: 75
150
145

3. Print list in reverse order using a loop.

Example:

Input: list1 = [10, 20, 30, 40, 50]

Output: 50
40
30
20
10

4. Print all prime numbers within a range.

Note: A Prime Number is a number that can only be divided by itself and 1 without remainders (e.g., 2, 3, 5, 7, 11)

A Prime Number is a natural number greater than 1 that cannot be made by multiplying other whole numbers.

Examples:

6 is not a prime number because it can be made by 2×3 = 6 37 is a prime number because no other whole numbers multiply to make it.

Input: # range start = 25 end = 50

Output: Prime numbers between 25 and 50 are:

29 31 37 41 43 47

5. Write a Python program to use for loop to find the factorial of a given number.

6. The factorial (symbol: !) means multiplying all numbers from the chosen number down to 1.

For example, a factorial of 5! is 5 × 4 × 3 × 2 × 1 = 120

Example:

Input: Enter an number : 5

Output: 120

7. Write a Python code to print multiplication table of any given number

Example:

Input: Enter a number: 2

Output: 2
4
6
8
10
12
14
16
18
20