Loops in Python

Loops in Python are used to execute a block of code repeatedly until a specified condition is met.

In Python, there are two primary types of loops: "for" and "while."

1. For Loop –
    - The for loop executes the block of code in it for each item in a sequence (list, string, or range).
    - The loop continues until all items in the sequence have been processed

    Syntax = for item in sequence:

    - **for**: The keyword that starts the loop.
    - **item**: The variable that takes the value of each element in the sequence on each iteration.
    - **in**: This keyword tells Python to loop through the sequence.
    - **sequence**: The collection (like a list, tuple, string, or range) that you're iterating over

Example 1: Iterating over a list	Example 2: Using range () for a sequence of numbers	Example 3: Iterating over a string

fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
print(fruit)
 	
Output:
apple
banana
cherry
	for i in range(5):
	# Iterates from 0 to 4
	print(i)
	
	Output –
	0
	1
	2
	3
	4
		for char in "hello":
		print(char)
		
		Output –
		h
		e
		l
		l
		o

2. While Loop –
   A while loop in Python repeatedly executes a block of code as long as a specified condition is True.

   Syntax – while condition:

   Example 1 –

```
count = 0

while count < 5:
    print("Count is:", count)
    count += 1 # Increment count by 1

print("Loop finished.")
```

Output -
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Loop finished.

Example 2 –

```
Number = 10
while(Number != 0) :
    print(Number)
    Number -=1
```

Output –
10
9
8
7
6
5
4
3
2
1

Note –

- **Infinite Loop**: If the condition never becomes False, the loop will run indefinitely. Make sure to update variables that affect the condition within the loop to avoid this.
- **break Statement**: The "break" statement is used to exit the loop prematurely. It can be applied to both "for" and "while" loops, allowing you to terminate the loop when a particular condition is met.
- **continue Statement**: The "continue" statement is used to skip the current iteration of the loop and proceed to the next one. It can be used in both "for" and "while" loops, enabling you to bypass certain iterations based on a condition.

Example of Infinite loop

```
num = 1

while True:  # Infinite loop
    print(f"Number: {num}")
    num += 1

    if num > 5:
        break  # Exit the loop when num exceeds 5
```

Output –

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

Explanation:

- The loop will run forever (while True) until the num exceeds 5.
- After printing the number, the condition if num > 5 checks if num is greater than 5, and if so, the break statement is executed to exit the loop.

Example of break statement

a)

```
num = 1

while num <= 5:  # Condition
    print(num)
    num += 1
```

Output:

```
1
2
3
4
5
```

b)

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    if number == 3:
        break
    print(number)
```

Output –
1
2

Example of continue statement

a)

```
num = 0
while num < 5:
    num += 1
    if num == 3:
        print("Skipping number 3")
        continue  # Skip the print statement for number 3
    print(f"Number: {num}")
```

Output –

Number: 1

Number: 2

Skipping number 3

Number: 4

Number: 5

Explanation:

- The loop runs while num is less than 5.
- When num equals 3, the continue statement skips the current iteration, so the print statement for num = 3 is skipped.

b)

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    if number == 3:
        continue
    print(number)
```

Output -
1
2
4
5

DevOps Example Automating Log File Analysis –

```
log_file = [
   "INFO: Operation successful",
   "ERROR: File not found",
   "DEBUG: Connection established",
   "ERROR: Database connection failed",
]
for line in log_file:
   if "ERROR" in line:
      print(line)
```

Output -

```
ERROR: File not found
ERROR: Database connection failed
```

Explanation - In this exercise, the loop iterates through the "log_file" list and prints lines containing the word "ERROR."