#### **1. Introduction to for Loop**

In Python, a for loop allows you to iterate over a sequence (like a list, tuple, or string) and perform actions for each element in the sequence. It's a fundamental concept for automating repetitive tasks in programming.

#### **2. Basic Syntax**

for item in iterable:

# Perform action with item

* **item**: Each element in the iterable (e.g., list, tuple).
* **iterable**: A sequence of elements (e.g., a list or string) that you want to iterate over.

#### **3. Iterating Over Sequences**

##### **a) List, Tuple, and String Iteration**

You can loop through lists, tuples, and strings to access each item.

# Loop through a list

my\_list = [1, 2, 3, 4, 5]

for item in my\_list:

print(item)

#### **4. Using range() with for Loop**

The range() function generates a sequence of numbers, which is useful for looping a specific number of times.

# Loop from 0 to 4

for i in range(5):

print(i)

# Loop from 1 to 5

for i in range(1, 6):

print(i)

# Loop with a step of 2

for i in range(0, 10, 2):

print(i)

#### **5. List Comprehension**

List comprehension is a concise way to create lists while using a for loop.

# Create a list of squares of numbers from 1 to 10

squares = [x\*\*2 for x in range(1, 11)]

print(squares)

#### **6. Enumerate - Get Index and Value**

The enumerate() function allows you to get both the index and the value of each item while iterating.

my\_list = ['a', 'b', 'c', 'd']

for index, value in enumerate(my\_list):

print(f"Index: {index}, Value: {value}")

#### **7. Using zip() to Iterate Over Multiple Sequences**

zip() allows you to loop through multiple sequences (like lists) at the same time.

names = ['Alice', 'Bob', 'Charlie']

scores = [90, 85, 88]

for name, score in zip(names, scores):

print(f"{name}: {score}")

#### **8. break and continue in Loops**

* **break**: Exits the loop when a condition is met.
* **continue**: Skips the current iteration and continues with the next one.

# Using `break`

for i in range(10):

if i == 5:

break # Exit loop when i equals 5

print(i)

# Using `continue`

for i in range(10):

if i == 5:

continue # Skip the iteration when i equals 5

print(i)

#### **9. Nested Loops (Loop Inside a Loop)**

You can use a for loop inside another for loop to handle multidimensional data.

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

for row in matrix:

for item in row:

print(item)

#### **10. Using else with for Loop**

The else block executes after the loop completes, unless a break statement interrupts the loop.

for i in range(1, 6):

print(i)

else:

print("Loop finished!")

#### **11. Conditional Statements in for Loop**

You can use if statements inside a for loop to execute code based on certain conditions.

# Print even numbers from 1 to 10

for i in range(1, 11):

if i % 2 == 0:

print(i)

#### **12. Using for Loop Inside Functions**

You can also use a for loop within a function to automate repetitive tasks.

def print\_elements(my\_list):

for element in my\_list:

print(element)

print\_elements([10, 20, 30, 40])

#### **Summary**

* **Basic usage**: for item in iterable
* **Using range()**: For looping over a sequence of numbers
* **List Comprehension**: Concise way to create lists
* **enumerate()**: Get both index and value
* **zip()**: Loop through multiple sequences simultaneously
* **Control Flow**: Using break and continue
* **Nested Loops**: Loops inside loops for multidimensional data
* **else in Loops**: Executes after the loop if no break occurs
* **Conditionals**: Use if inside loops to control flow