

Lab Manual: Loops in Python

Objective:

By the end of this lab, you will understand the different types of loops in Python, how to use them to perform repetitive tasks, and how to control loop flow with statements like `break`, `continue`, and `else`. This lab will provide real-world examples and an assignment to reinforce learning.

Introduction to Loops in Python:

Loops are used to execute a block of code repeatedly until a certain condition is met. Python provides two primary types of loops:

1. **for loops:** Used to iterate over a sequence (such as lists, tuples, strings).
2. **while loops:** Used to repeat a block of code as long as a condition is true.

Part 1: The `for` Loop

Objective:

- Learn how to use a `for` loop to iterate over a sequence such as a list, string, or range of numbers.

Task 1: Iterating Over a List

Example:

```
fruits = ["apple", "banana", "cherry"]

# Use a for loop to iterate through each element in the list
for fruit in fruits:
    print(fruit)
```

Explanation:

- The `for` loop iterates over each element in the list `fruits` and prints each one.

Task 2: Using `range()` in a `for` Loop

Example:

```
# Use a for loop with range to iterate through a sequence of numbers
for i in range(5): # Iterates from 0 to 4
    print(i)
```

Explanation:

- The `range(5)` generates numbers from 0 to 4. The loop will execute the print statement 5 times.

Task 3: Iterating Over a String

Example:

```
name = "Python"

# Use a for loop to iterate over characters in a string
for letter in name:
    print(letter)
```

Explanation:

- The `for` loop iterates over each character in the string `name` and prints it.

Assignment 1:

1. Write a program that takes a list of numbers and prints the **square** of each number using a `for` loop.
 - **Hint:** Use `for number in numbers_list: print(number**2).`
2. Write a program that uses the `range()` function to print all **even numbers** from 1 to 20.

Part 2: The `while` Loop

Objective:

- Learn how to use a `while` loop to execute a block of code as long as a specified condition is true.

Task 1: Basic `while` Loop

Example:

```
count = 0

# Use a while loop to print numbers 0 to 4
while count < 5:
    print("Count is:", count)
    count += 1
```

Explanation:

- The `while` loop continues as long as the condition `count < 5` is true. The variable `count` is incremented by 1 in each iteration.

Task 2: Infinite Loop with `break`

Example:

```
while True:
    user_input = input("Type 'exit' to stop: ")
    if user_input == 'exit':
        break # Exit the loop
```

Explanation:

- This creates an **infinite loop** that only stops when the user types 'exit'. The `break` statement terminates the loop.

Task 3: Using `else` with a `while` Loop

Example:

```
python
Copy code
counter = 0

while counter < 3:
    print("Counter is:", counter)
    counter += 1
else:
    print("Loop completed successfully")
```

Explanation:

- The `else` block is executed when the `while` loop condition becomes false.

Assignment 2:

1. Write a Python program that uses a `while` loop to print the numbers from 1 to 10.
2. Write a Python program that continuously asks the user to enter a number. The loop should exit when the user enters a negative number, and print the sum of all the entered numbers.

Part 3: Controlling Loop Execution with `break`, `continue`, and `else`

Objective:

- Learn how to control loop flow using `break`, `continue`, and `else` statements.

Task 1: Using `break` in a `for` Loop

Example:

```
for i in range(1, 10):
    if i == 5:
        break # Exit the loop when i equals 5
    print(i)
```

Explanation:

- The loop breaks when `i` equals 5, so it only prints numbers from 1 to 4.

Task 2: Using `continue` to Skip an Iteration

Example:

```
for i in range(1, 10):
    if i % 2 == 0:
        continue # Skip even numbers
    print(i)
```

Explanation:

- The loop skips printing even numbers because of the `continue` statement.

Task 3: Using `else` with `for` Loops

Example:

```
python
Copy code
for i in range(5):
    print(i)
else:
    print("Loop finished without breaking")
```

Explanation:

- The `else` block is executed after the loop finishes normally (without encountering a `break`).

Assignment 3:

1. Write a program that uses a `for` loop to print numbers from 1 to 20, but skips multiples of 3 using the `continue` statement.

2. Write a program that asks the user to guess a number (between 1 and 10). If the guess is correct, the program prints "Correct guess!" and exits the loop. If the guess is wrong, the loop continues until the correct number is guessed.

Part 4: Nested Loops

Objective:

- Learn how to use loops inside other loops, also known as **nested loops**.

Task 1: Multiplication Table with Nested Loops

Example:

```
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:

- This prints a multiplication table where the outer loop controls the rows, and the inner loop controls the columns.

Task 2: Printing a Pattern

Example:

```
n = 5
for i in range(1, n + 1):
    for j in range(1, i + 1):
        print("*", end="")
    print()
```

Explanation:

- This prints a right-angle triangle pattern using nested loops.

Assignment 4:

1. Write a program to print the multiplication table from 1 to 10 using nested loops.
2. Write a program that prints the following pattern using nested loops:

```
1
22
333
4444
55555
```