Loops 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). Additionally, Nested Loops allow looping within loops for more complex tasks. While all the ways provide similar basic functionality, they differ in their syntax and condition-checking time

1. ***For Loop in Python :*** For loops are used for sequential traversal.

   Syntax:

   For variable_name in range(start,end,step):

   statements(s)

   ←----------→

```
n = 4
for i in range(0, n):
   print(i)
```

   Output

```
0
1
2
3
```

   ←----------→

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
   print(fruit)
```

2. ***While loop***

   Used to run a block of code as long as a condition is `True`.

   Syntax:

```
while condition:
   # code block
```

   ←----------→

```
i = 0
while i < 5:
   print(i)
   i += 1
```

   Output:

```
0
```

1
2
3
4

3. ***Loop Control Statements:***

Statement	Purpose
`break`	Stops the loop entirely
`continue`	Skips the current iteration and moves to the next
`pass`	Does nothing (used as a placeholder)

Example using break:

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

Example using continue:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Practice Questions:

a) Find sum of digit (like a=123 then 1+2+3=6)
b) Reverse a string like(a=”abcd” then a=”dcba”
c) Check whether string is palindrome or not (like a=”abcba” then reverse of a is also same.)

1. Simple Right-Angled Triangle

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

```
*
**
***
****
*****
```

2. Inverted Triangle

```
n = 5
for i in range(n, 0, -1):
    print('*' * i)
```

Output:

```
*****
****
***
**
*
# 1
# 22
# 333
# 4444
# 55555

# rows=int(input("enter"))
# for i in range(rows+1):
#     for j in range(i):
#         print(i,end='')
#     print('')

# 55555
# 5555
# 555
# 55
# 5

# rows=5
# num=rows
# for i in range(rows,0,-1):
#     for j in range(0,i):
#         print(num,end='')
#     print('')


# *
# **
# ***
# ****
# *****

# rows=5
# for i in range(0,rows):
#     for j in range(0,i+1):
#         print("*",end='')
#     print('')
```

What is a Function in Python?

A **function** in Python is a **block of reusable code** that performs a specific task. It allows you to group a set of instructions under a name and execute them when needed — making your code more **modular**, **organized**, and **reusable**.

Why Use Functions?

- **Avoid repetition**: Write once, use many times.
- **Code organization**: Makes code easier to understand and debug.
- **Reusability**: Reuse logic across different parts of a program.
- **Testing**: Easier to test small components individually.

Function Syntax in Python

```
def function_name(parameters):
        """optional docstring"""
        # code block
        return result
```

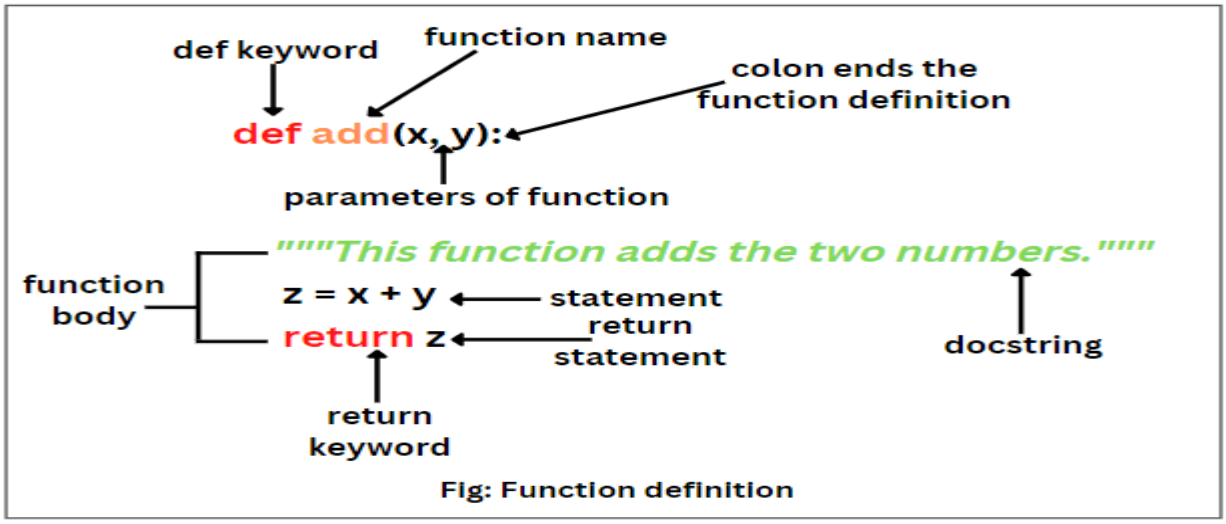

Fig: Function definition

Let's Break Down the Syntax

Component	Description
def	Keyword to define a function
function_name	Any valid identifier (name of the function)
parameters	Optional: values passed into the function
:	Indicates the start of the function body
return	Optional: returns a value from the function
docstring	Optional: explains what the function does

Example:

```
def greet(name):
   print("Hello", name)
greet("cse")
```

Types of Functions in Python

1. **Built-in Functions**
   Example: len(), print(), type(), etc.
2. **User-defined Functions**
   Functions you create using def.
3. **Lambda Functions (Anonymous)**
   Short functions created with the lambda keyword.

4. **Recursive Functions**
   A function that calls itself to solve problems like factorial, Fibonacci, etc.
5. **Generator Functions**
   Use yield instead of return to produce a sequence of values lazily.

What is a `lambda` Function?

A **lambda function** in Python is a **small anonymous function** (i.e., without a name), defined using the lambda keyword. It's useful for **short operations** that are used once or for functional programming (like with map, filter, reduce).

Syntax of a Lambda Function

lambda arguments: expression

Basic Lambda

```
square = lambda x: x ** 2

print(square(5))
```

Add Two Numbers

```
add = lambda a, b: a + b

print(add(3, 4))
```

Normal Function vs Lambda Function

Feature	**Normal Function**	**Lambda Function**
Definition	Using def keyword	Using lambda keyword
Name	Has a name (e.g., def add():)	Usually anonymous (can be assigned to a variable)
Multi-line	Can span multiple lines	Only one expression allowed
Statements allowed	Yes (loops, conditions, print, etc.)	No — only expressions allowed
Use Case	Complex or reusable logic	Short-term, one-off use, usually in-line
Return behavior	Must use return keyword	Automatically returns the result
Readability	More readable for large tasks	Good for very simple logic

Practice:

Find the maximum of two numbers

```
maximum = lambda a, b: a if a > b else b        print(maximum(10, 20))
```

```
                                    if a > b:
                                        return a
                                    else:
def maximum(a, b):                      return b
```

What is a Recursive Function?

A **recursive function** is a function that **calls itself** to solve a problem by breaking it down into smaller sub-problems.

It typically has:

- A **base case** – to stop recursion
- A **recursive case** – the function calls itself with smaller inputs

Syntax of a Recursive Function

```
def function_name(parameters):
    if base_case_condition:
        return base_case_value
    else:
        return function_name(smaller_input)
```

Factorial using Recursion

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
# Output: 120
```

How it works:

```
factorial(5)
= 5 * factorial(4)
= 5 * 4 * factorial(3)
= 5 * 4 * 3 * factorial(2)
= 5 * 4 * 3 * 2 * factorial(1)
= 5 * 4 * 3 * 2 * 1
= 120
```

Practice

Fibonacci Series (Nth Term)

```
def fibonacci(n):
    if n == 0:
```

```
        return 0
    elif n == 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
print(fibonacci(6))
```

Questions:

1. Create function Count Vowels in a String
2. Create function Check Prime
3. Create function Reverse a Number
4. Create function to Armstrong Number?
   An **Armstrong number** (or **narcissistic number**) of n digits is a number such that the **sum of its digits raised to the power n** equals the number itself.
   Examples:

- 153 → 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153 ✅
- 9474 → 9^4 + 4^4 + 7^4 + 4^4 = 9474 ✅
- 123 → 1^3 + 2^3 + 3^3 =36 ❌

5. Create function Duck number.
   What is a Duck Number?
   A **Duck Number** is a **number that contains at least one zero, but not at the beginning.**

✅ **Examples:**

- 3210 → ✅ Duck (has a zero, not at the start)
- 703 → ✅ Duck
- 0123 → ❌ Not a Duck (starts with zero — and also not valid as an integer in Python)
- 123 → ❌ Not a Duck (no zero at all)

6. Create function to find Spy Number?

A **Spy Number** is a number where the **sum of its digits is equal to the product** of its digits.

🔍 **Examples:**

1124 → ✅ Spy

Sum = 1+1+2+4 = 8

Product = 1×1×2×4 = 8

123 → ❌ Not a Spy

Sum = 6, Product = 6 → ✅

(Correction: 123 **is** Spy!)

134 → ❌ Not a Spy

Sum = 1+3+4 = 8, Product = 1×3×4 = 12

- wap to calculate the sum of all the numbers from 1 to given number
- wap to print the following pattern

1

12

123

1234

12345

- wap to print multiplication table of given number
- wap to count total number of digits in a given number. *(using while loop)*
- wap to print a list in reverse order by using loop
  [10,20,50,60,40,80,40,90,70,20,10,30] *(sort in ascending order first)*
- wap to display all the prime number within a range
  6 is not a prime number because it can be made by 2x3 = 6
  37 is a prime number because no other whole numbers multiply together to make it
- wap to display fibonacci series up to 10 terms
  The Fibonacci Sequence is a series of numbers. The next number is found by adding up the two numbers before it.
  The first two numbers are 0 and 1.
  For example, 0, 1, 1, 2, 3, 5, 8, 13, 21. The next number in this series above is 13+21 = 34.
- wap to find the factorial of given number