Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 26th February 2024
Semester: 2nd	**Session:** 2023
Lab/Project/Assignment #: 6	**CLOs to be covered:** CLO 2
Lab Title: Recursion in Python	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO 2	Practice collaboratively on large problems and provide their working solutions.					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**9-10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**7-8**	L2	Submitted the lab tasks but have good understanding
Good	**5-6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**3-4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**1-2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 6

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of Recursion in Python
- To Become familiar with the working of recursion in Python
- To learn about multiple types of recursions in Python
- To practice some examples of recursion in Python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

Recursion in Python

- Recursion is a technique in computer programming where a function calls itself repeatedly until it reaches a base case, at which point the function returns the result of the base case. Python supports recursion and it is a powerful technique that can be used to solve complex problems in a more concise and elegant way than traditional iterative approaches.

- The basic idea behind recursion is to break down a problem into smaller sub problems and solve each sub-problem using the same approach. This process continues until the sub-problems are simple enough to be solved directly, which is known as the base case. Once the base case is reached, the function returns the result and the recursive calls are unwound in a Last-In-First-Out (LIFO) order until the original call is reached.

Practical Example # 1

Imagine you want to know the name of a person in the queue that you are standing in. You ask people to find out about it.

They keep asking the next person until they find an answer. Once an answer is found, they send it back until it reaches you.

The above two examples depict a process in which a particular action is repeated until a condition is met. These processes have a strong resemblance to what we call recursion.

A process in which a function calls itself is called recursion. This process helps ease the method of solving problems by replacing iterative code with recursive statements.

Practical Example # 2

Imagine sitting in an auditorium and being confused about the row you are sitting in. Since you cannot count till the first bench, you ask the person right before you what row they are sitting in. If the person doesn't know his seat number, he will ask the person right before him.

The process of asking will continue till someone knows their seat number. When the person in the 3rd row informs that he is sitting in row 3, the person behind him will add one and pass on his row number 4 to the person right

after. The row number will be calculated in each row and will be passed on until it reaches you. Once it reaches you, add 1 to get your final row number.

That is precisely how recursion in Python works.

Syntax:

```
def rec_func_name():
  if(condition):                # base case
    simple statement without recursion
  else:                         # general case
    statement calling rec_func_name()
```

Base Case:

This helps us to terminate the recursive function. It is a simple case that can be answered directly and doesn't use recursion. If satisfied, it returns the final computable answer. If this is omitted, the function will run till infinity.

General (Recursive) Case:

This case uses recursion and is called unless the base condition is satisfied.

The base case and its solution must first be identified to write a recursive function. More than one base case can be used depending on the situation. Once a base case has been identified, the general or recursive case has to be identified so that each recursive call takes us one step closer to achieving the base case.

A recursive function makes use of a call stack. Every time the recursive function gets called, that function gets added to the top of this stack. Imagine opening up an onion and you place every layer you peel near it. So, each layer, peeled, would be placed at the top of this peel stack.

Now compare this to a recursive function. Peeling each layer would be a recursive function call. When the first layer is peeled, the first peel() would be added at the top of the stack, then the next would be added above it and so on, until the process is completed.

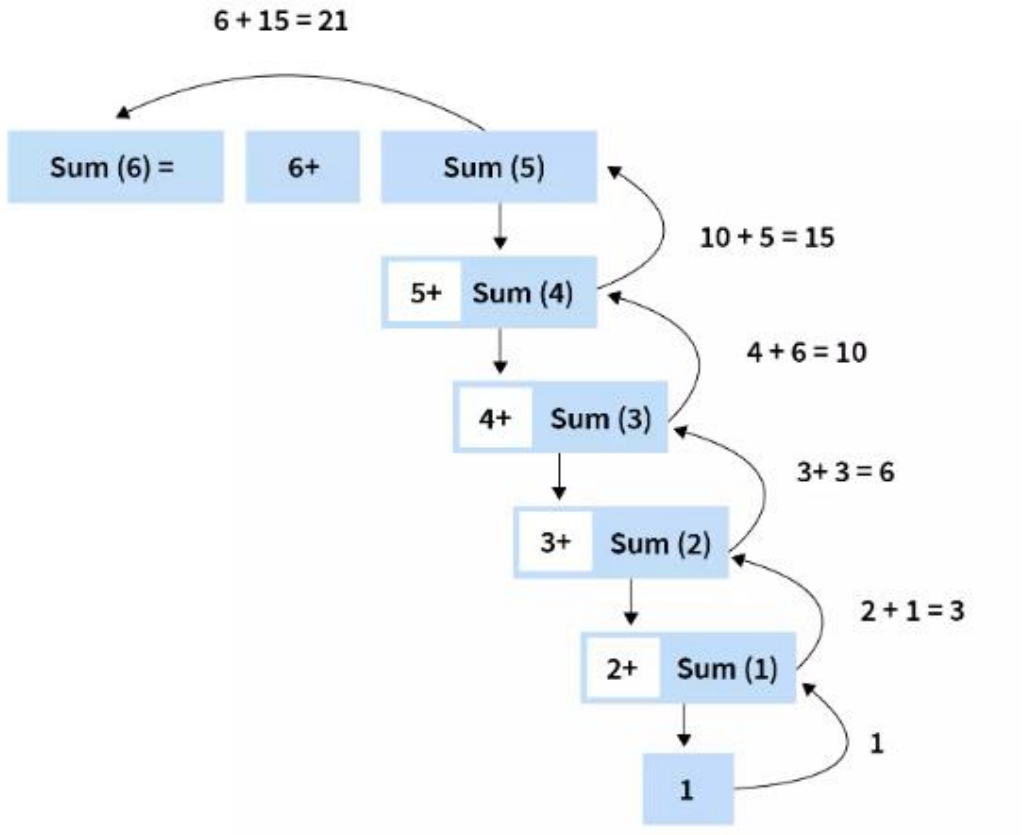
Example:

Suppose you want to find the sum of n natural numbers in python.

```
def sum(n):
   if n <= 1:   # base case
      return n
   else:       # general or recursive case
   return n + sum(n - 1)
print(sum(6))
```

When we call the function sum() with any positive integer, say 6, it will recursively call itself by decreasing the number. As shown in the recursive case, each function call adds the number argument with the value of the sum of the decremented number.

Let's understand this better through the step-by-step process diagram below. The recursive process starts when sum(6) is called and ends when we move into the base case, i.e., when sum (1) returns 1. Once this value is returned, it is added to all the values in the call stack as shown and finally, the value 21 is returned.



All recursive functions have two distinct phases of working - the **winding** and **unwinding** phase.

The winding phase embarks with the recursive function being called the first time and moves ahead until the last recursive call. In this phase, no return statements are executed. The winding phase terminates when the base case's condition becomes true in a call.

That's when the unwinding phase begins. In this phase, all recursive functions return in the reverse order till the first instance of the function returns.

Types of Recursions in Python

- Direct Recursion
- Indirect Recursion

Direct Recursion:

In this type of recursion, the function calls itself. It also has two types.

Tail Recursion

A recursive call is said to be tail-recursive if it is the last statement to be executed inside the function.

Example:

```
def disp(n):
  if n == 0:
    return  # Base case
  print(n)
  disp(n - 1)  # Tail Recursive Call

disp(5)
```

In the first invocation of the function, the print statement causes 5 to be displayed on the screen and disp(4) is added at the top of the stack. In the next invocation, 4 is printed and disp(3) is added to the stack.

Similarly, 3, 2, and 1 are printed in the following invocations. In the last call, the value of n becomes 0, and thus the base case is satisfied, causing recursion to terminate.

Head Recursion

If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head Recursion. There's no statement, no operation before the call.

Example:

```
def disp(n):
    if n == 0:
        return  # Base case
    disp(n - 1)  # Not a Tail Recursive Call
    print(n)
disp(5)
```

At consecutive invocations of the function, disp(5), disp (4), disp(3), disp(2), disp (1) are added to the top of the stack, respectively. When we encounter disp(0), the terminating condition is satisfied, returning the function. At this point, the unwinding phase begins. The calls return in reverse order, with disp(1) returning first, followed by the rest.

Thus, the print statement of disp(1) occurs before. So, the values are printed in ascending order.

Indirect Recursion:

In this type of recursion, the function calls another function which calls the original function

Example:

```
def A(n):
    if n > 0:
        print("", n, end='')
        B(n + 1)

def B(n):
    if n > 1:
        print("", n, end='')
        A(n - 5)
A(20)
```

When function A() is called, it first executes the print statement and then calls function B() with an incremented value of n. Its print statement is executed within function B, and then the function A() with a value of n reduced by five is called.

The process continues as long as the terminating condition is not satisfied.

Note: The chain of functions in indirect recursion may involve any number of functions. We can have f1() call f2() that calls f3() that calls f4() that calls f5() that in turn calls f1(). The usage of indirect recursion increases the complexity of code and is thus not very common.

Lab Tasks

- Write the python program to calculate the Permutation and Combination of **n** and **r** using recursion.
- Write the python program to print the first 20 terms of the Fibonacci sequence by using recursion.

  Hint: In mathematics, the Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones.

  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …….
- Write a python program that make a list of 10 random numbers ranging from 0 to 1000, and add all these random numbers by using recursion
- Write a python program to calculate power of a number by using recursion, and takes that number and its power from the user.
- Write a python program that takes a string from the user and return the inverted string by using recursion.
- Write a python program that has a list which contain some strings, integers, lists, tuples and sets. Make a new list by using recursion that contains only string values in it.