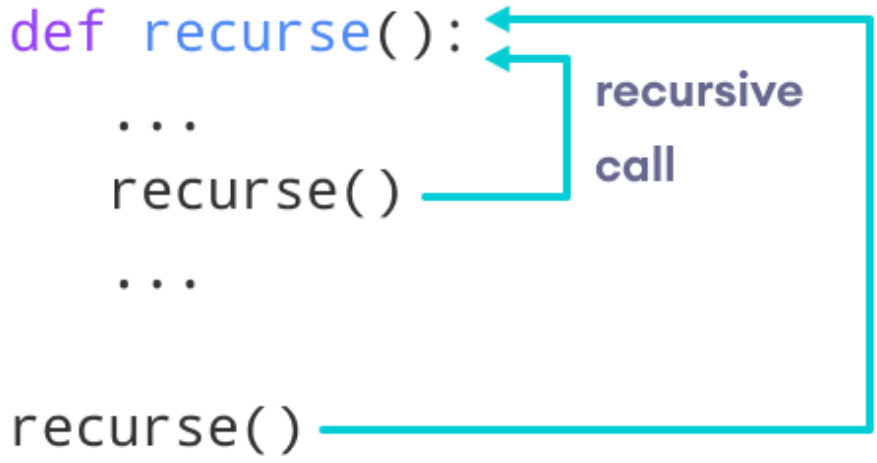
Recursion: In recursion, a function calls itself repeatedly. This technique is generally used when a problem can be divided into smaller subproblems of the same form.

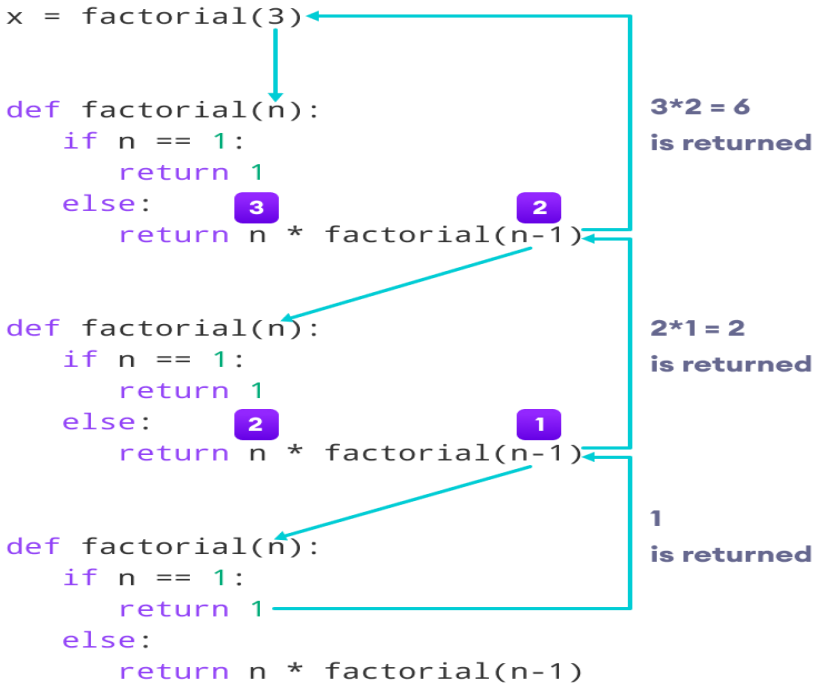


Example of a recursive function

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("The factorial of", num, "is", factorial(num))
```

Let's look at an image that shows a step-by-step process of what is going on:



Q 1. Explain the step by step working of this code and predict the output:

```
def recursion(k):
  if(k > 0):
    result = k + recursion(k - 1)
    print(result)
  else:
    result = 0
  return result
print("Recursion Example Results")
recursion(3)
```

Sol.

```
Recursion Example Results
1
3
6
```

Q2. Explain the step by step working of this code and predict the output:

```
total = 0
def sum_nestedlist( l ):
    global total
    for j in range(len(l)):
        if type(l[j]) == list :
            sum_nestedlist(l[j])
        else:
            total += l[j]
sum_nestedlist([[1,2,3],[4,[5,6]],7])
print(total)
```

Sol. 28

Q3. Explain the step by step working of this code and predict the output:

```
def P(n, x):
    if(n == 0):
        return 1
    elif(n == 1):
        return x
    else:
        return (P(n-1, x)+(n-1)*P(n-2, x))
n = 3
X = 5
print(P(n, X))
```

Sol. 16

Q4. Explain the step by step working of this code and predict the output:

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])
        line += [1]
    return line
print(pascal(6))
```

Sol. [1, 5, 10, 10, 5, 1]

Q5. Explain the step by step working of this code and predict the output:

```
def mult3(n):
    if n == 1:
        return 3
    else:
        return mult3(n-1) + 3

for i in range(1,10):
    print(mult3(i))
```

Sol.

```
3
6
9
12
15
18
21
24
27
```

Q6. Explain the step by step working of this code and predict the output:

```
def printSubsequences(arr, index, subarr):
    if index == len(arr):
        if len(subarr) != 0:
            print(subarr)
    else:
        printSubsequences(arr, index + 1, subarr)

        printSubsequences(arr, index + 1,
                          subarr+[arr[index]])
    return
arr = [1, 2, 3]
printSubsequences(arr, 0, [])
```

Sol.

```
[3]
[2]
[2, 3]
[1]
[1, 3]
[1, 2]
[1, 2, 3]
```

Q7. Explain the step by step working of this code and predict the output:

```
def power(N, P):
    if(P == 0 or P == 1):
        return N
    else:
        return (N*power(N, P-1))
N = 5
P = 2
print(power(N, P))
```

Sol. 25

Q8. Explain the step by step working of this code and predict the output:

```
def flattenList(nestedList):
    if not(bool(nestedList)):
        return nestedList
    if isinstance(nestedList[0], list):
        return flattenList(*nestedList[:1]) + flattenList(nestedList[1:])
    return nestedList[:1] + flattenList(nestedList[1:])
nestedList = [[8, 9], [10, 11, 'geeks'], [13]]
print('Nested List:\n', nestedList)
print("Flattened List:\n", flattenList(nestedList))
```

Sol.

Nested List:
[[8, 9], [10, 11, 'geeks'], [13]]
Flattened List:
[8, 9, 10, 11, 'geeks', 13]

Q9. Given a List of positive integers. We need to make the given list a 'Palindrome'. Only allowed operation on list is merge. Merging two adjacent elements means replacing them with their sum. The task is to find minimum number of merge operations required to make given list a 'Palindrome'.

Sample Input/Output :

Test Case 1:
Input : arr[] = {15, 4, 15}
Output : 0
Explanation: Array is already a palindrome. So, we do not need any merge operation.

Test Case 2:
Input : arr[] = {1, 4, 5, 1}
Output : 1
Explanation: We can make given array palindrome with minimum one merging (merging 4 and 5 to make 9)

Test Case 3:
Input : arr[] = {11, 14, 15, 99}
Output : 3
Explanation: We need to merge all elements to make a palindrome.

Solution.

```
def findMinOps(arr, n):
    ans = 0 #
    i,j = 0,n-1
    while i<=j:
        if arr[i] == arr[j]:
            i += 1
            j -= 1
        elif arr[i] > arr[j]:
            j -= 1
            arr[j] += arr[j+1]
            ans += 1
        else:
            i += 1
            arr[i] += arr[i-1]
            ans += 1
    return ans
arr = [1, 4, 5, 9, 1]
n = len(arr)
print(arr)
print("Count of minimum operations is " + str(findMinOps(arr, n)))
```

Q10. Given a sorted list (sorted in non-decreasing order) of positive numbers, find the smallest positive integer value that cannot be represented as sum of elements of any subset of given list.

Sample Input/Output:

Test Case 1:
Input: List = [1, 3, 6, 10, 11, 15]
Output: 2

Test Case 2:
Input: List = [1, 1, 1, 1]
Output: 5

Test Case 3:
Input: List = [1, 1, 3, 4]
Output: 10

Test Case 4:
Input: List = [1, 2, 5, 10, 20, 40]
Output: 4

Recursion Functions in Python

Sol.

```
def findSmallest(arr, n):
    res = 1
    for i in range (0, n ):
        if arr[i] <= res:
            res = res + arr[i]
        else:
            break
    return res
List= [1, 1, 1, 1]
n1 = len(List)
print(findSmallest(List, n1))
```