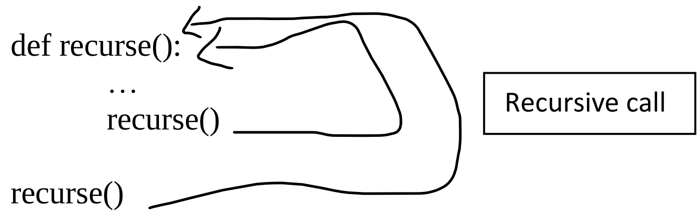
Function Recursion:

1. Recursion is the process of defining something in terms of itself.
2. When a function calls itself, it is known as recursion.
3. A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.



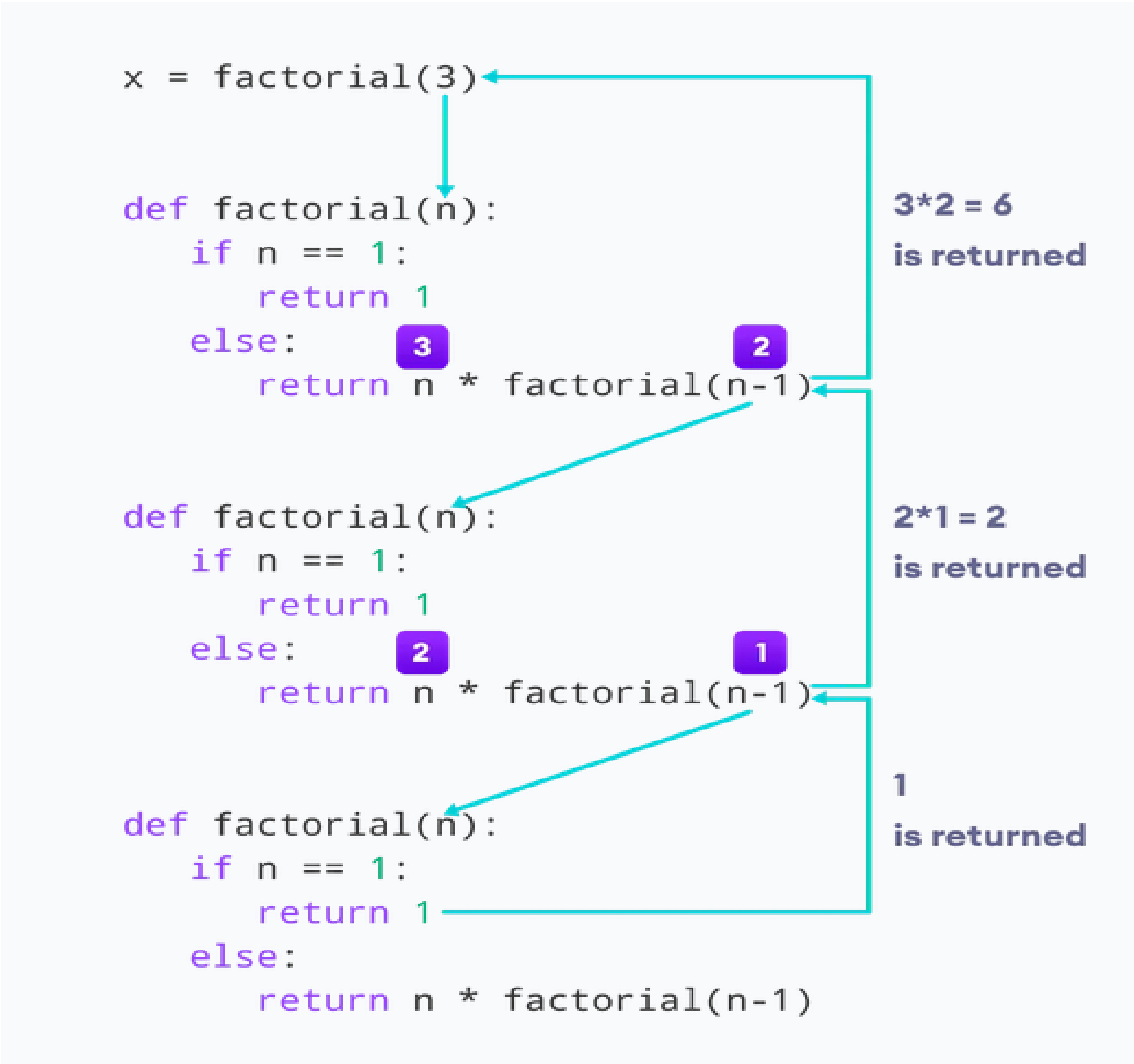
Example of recursive function (Program of Factorial):

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

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


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

Recursive call:

```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Working:



Advantages:

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages:

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Tail Recursion:

1. A unique type of recursion where the last procedure of a function is a recursive call.
2. The recursion may be automated away by performing the request in the current stack frame and returning the output instead of generating a new stack frame.
3. The tail-recursion may be optimized by the compiler which makes it better than non-tail recursive functions.

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

```
def fun1(k):
  if(k > 0):
    result = k + fun1(k - 1)
    print(result)
  else:
    result = 0
  return result
print("Results")
recursion(4)
```

Sol.

```
Results
1
3
6
10
```

Q2. What will be the output of the following code?

```
def fun_recursive(n):
    print("Calculating F", "(", n, ")", sep="", end=", ")

    # Base case
    if n == 0:
        return 0
    elif n == 1:
```

```
        return 1

    # Recursive case
    else:
        return fun_recursive (n-1) + fun_recursive(n-2)
fun_recursive(2)
```

Sol.

```
Calculating F(2), Calculating F(1), Calculating F(0)
1
```

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

```
sum = 0
def list1(lst):
    global sum
    for j in range(len(lst)):
        if type(lst [j]) == list:
            list1(lst[j])
        else:
            sum += lst[j]
list1([[11,12,13],[14,[15,16]],17])
print(sum)
```

Sol.

```
98
```

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

```
def printPattern(targetNumber) :

  # Base Case
  if (targetNumber <= 0) :
    print(targetNumber)
    return

 # Recursive Case
  print(targetNumber)
  printPattern(targetNumber - 5)
```

```
  print(targetNumber)

# Driver Program
n = 10
printPattern(n)
```

Sol.

10
5
0
5
10

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

```
def fun2(n, x):
    if(n == 1):
        return x
    elif(n == 0):
        return 1
    else:
        return (fun2 (n-1, x)+(n-1)* fun2(n-2, x))
n = 4
X = 4
print(fun2(n, X))
```

Sol.

28

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

```
def fun1(array):
  if len(array) == 0:
    return []
  elif len(array) == 1:
   return array
  return [array[len(array) - 1]] + fun1(array[:len(array) - 1])
# Driver Code
array = [1, 2, 3, 4]
print(fun1(array))
```

```
Sol.
    [4, 3, 2, 1]
```

Q7. Think of a recursive version of the function f(n) = 3 * n,
i.e., the multiples of 3

Mathematically, we can write it like this:

f(1) = 3,
f(n+1) = f(n) +3,

Sol.

A Python function can be written like this

```
def mult3(n):
    if n == 1:
        return 3
    else:
        return mult3(n-1) + 3
for i in range(1,10):
        print(mult3(i))
```

Q8. Think of a recursive version of the Fibonacci series up to n terms

Sol.

```
def recursive_fibonacci(n):
    if n <= 1:
        return n
    else:
        return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2))

n_terms = 7

# check if the number of terms is valid
if n_terms <= 0:
    print("Invalid input ! Please input a positive value")
else:
    print("Fibonacci series:")
for i in range(n_terms):
    print(recursive_fibonacci(i))
```

Q9. Think of a recursive version to calculate the sum of the positive integers of n+(n-2)+(n-4)... (until n-x =< 0).

input:	**output:**
sum_series(6)	12
sum_series(10)	30

Sol.

```
def sum_series(n):
  if n < 1:
    return 0
  else:
    return n + sum_series(n - 2)

print(sum_series(6))
print(sum_series(10))
```

Q10. For the given list having positive integers, and elements of the list are sorted in non-decreasing. Find the smallest positive integer value that cannot be represented as the sum of elements of any subset of the given list.

Sample Input/Output:

Test Case 1:
Input: List = [1, 2, 4, 11]
Output: 8

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, 4, 11, 12, 99]
Output: 8

Sol.

```
def findSmallest(arr, n):
    ele = 1
    for i in range (0, n ):
        if arr[i] <= ele:
            ele = ele + arr[i]
        else:
            break
    return ele
List= [1, 2, 4, 11, 12, 99]
n1 = len(List)
print(find_Smallest_ele(List, n1))
```