Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

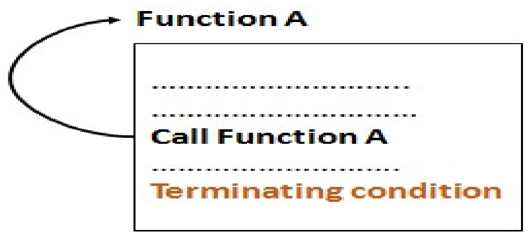
Lecture #59
Recursion

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Introduction

Recursion is a process where a function calls itself to produce a result with a terminating condition included within it. Some algorithms are very easy to write using the recursive paradigm, while others are not. There is no recursive function that cannot be rewritten in an iterative fashion. So it's usually up to the programmer to choose the best approach for the case at hand.



Criteria for Recursion:

- **Recursive case:** The recursive case is the part where the function calls on itself .
- **Base case or Termination case:** This the condition that stops the recursion.

Need of Recursion:

- It can simplify the code by reducing code length.
- Complex problems can be broken down into simpler sub programs.
- Sequence generation is easier with recursion than using some nested iteration.

Characteristics of Recursive function:

- There must be **at least one base case** for which the solution is known without further recursive breakdown.
- Problems that are not a base case are broken down into sub problems and work towards a base case.
- There is a way to derive the solution of the original problem from the solutions of the recursively solved sub problems.

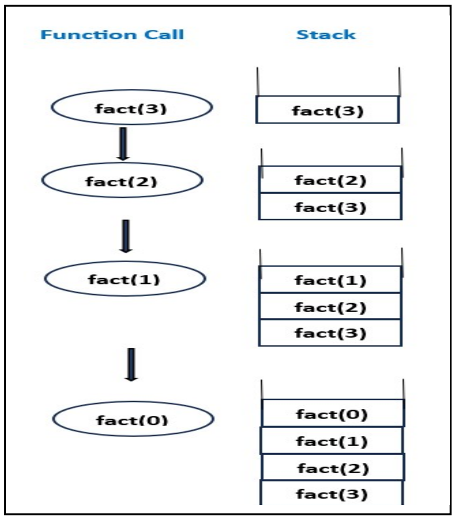
Implementation of Recursion

Recursion is **implemented using stack** because activation records are to be stored in LIFO order (last in first out). **An activation record of a function call contains arguments, return address and local variables of the function.**

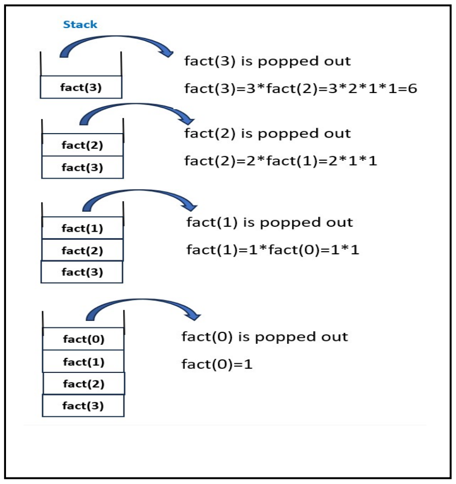
Stack is a linear data structure in which elements are inserted to the top and deleted from the top.

Consider an example of computing the factorial of a given number using recursion. Observe the activation record stored in the stack for each call.

```
def factorial(n):
        if  n==0:
                return  1
        else:
                return (n*factorial(n-1))
n = 3
print(factorial(n))
```



When the function call is made recursively, the activation record for each call will be placed on the top of the stack. Initially, fact(3) is called which recursively calls fact(2), fact(1),fact(0) and activation record gets inserted to top of the stack.



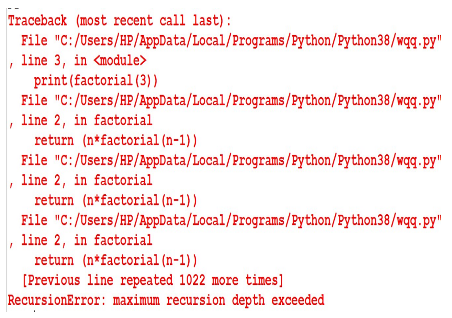
For n=0 i.e. base case(Termination Condition), the function call stops and fact(0) is popped out from the top of the stack. It returns 1, then the recursion backtracks and solve the pending function calls are popped out of the stack and fact(3) is computed.

Stack Memory Allocation

In Python, function calls and the references are stored in **stack memory.** Allocation happens on contiguous blocks of memory – referred as ***Function call Stack***. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack. Any local memory assignments such as variable initializations inside the particular functions are stored temporarily on the function call stack, where it is deleted once the function returns. This allocation onto a contiguous block of memory is handled by the compiler using predefined routines.

RecursionError: Recursive function without a termination condition.

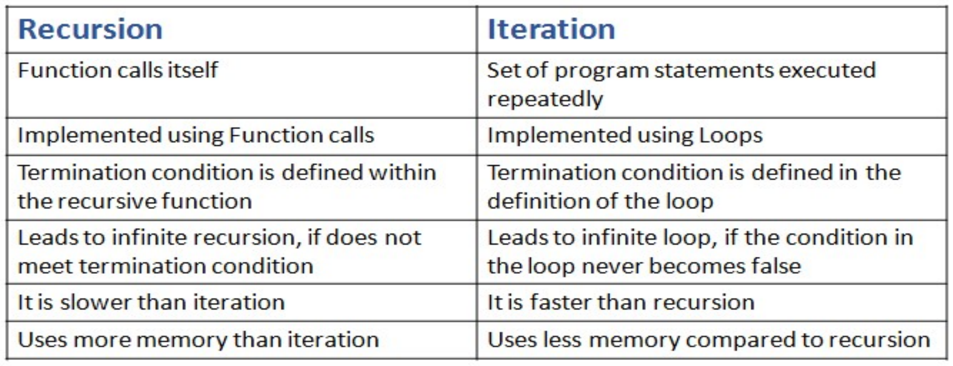
```
def factorial(n):
        return (n*factorial(n-1))
print(factorial(3))
```



```
Traceback (most recent call last):
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/wqq.py"
, line 3, in <module>
    print(factorial(3))
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/wqq.py"
, line 2, in factorial
    return (n*factorial(n-1))
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/wqq.py"
, line 2, in factorial
    return (n*factorial(n-1))
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/wqq.py"
, line 2, in factorial
    return (n*factorial(n-1))
  [Previous line repeated 1022 more times]
RecursionError: maximum recursion depth exceeded
```

In the above as we have not entered the termination condition the factorial function will recursively called for maximum number of times (nearly 1022 times) and afterwards it will display recursion error and quits from execution. So, to prevent the program termination with recursion error we need to specify the termination criteria for recursive function.

Difference between Recursion and Iteration

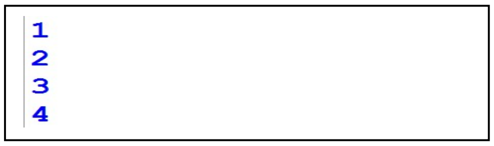


Recursion	Iteration
Function calls itself	Set of program statements executed repeatedly
Implemented using Function calls	Implemented using Loops
Termination condition is defined within the recursive function	Termination condition is defined in the definition of the loop
Leads to infinite recursion, if does not meet termination condition	Leads to infinite loop, if the condition in the loop never becomes false
It is slower than iteration	It is faster than recursion
Uses more memory than iteration	Uses less memory compared to recursion

Print all the numbers from 1 to 4.

Using Iteration:

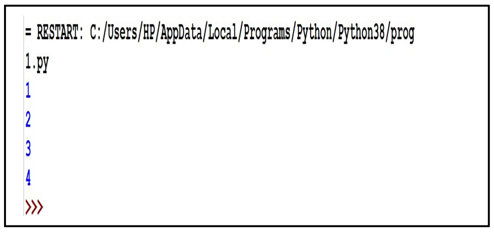
```
for i in range(1, 5):
    print(i)
```



```
1
2
3
4
```

Using Recursion:

```
def  print_numbers(n):
            if n == 5:
                    return 1
            print(n)
             print_numbers(n + 1)
print_numbers(1)
```



```
= RESTART: C:/Users/HP/AppData/Local/Programs/Python/Python38/prog
1.py
1
2
3
4
>>>
```

Note: When a problem can be solved both recursively and iteratively with similar programming effort, it is generally best to use an iterative approach. Recursion is often used to solve problems that are hierarchical in nature, such as tree traversal and sorting. Iteration is often used to solve problems that are sequential in nature, such as searching and filtering.

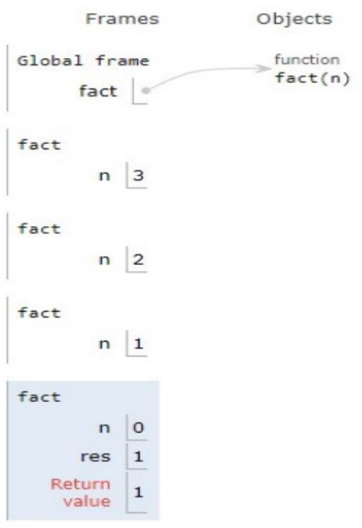
Advantages of Recursion:

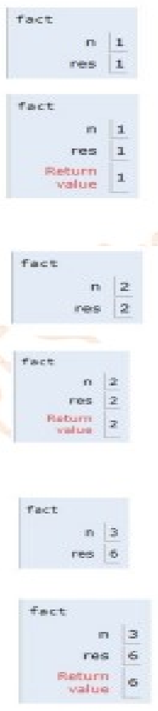
- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion:

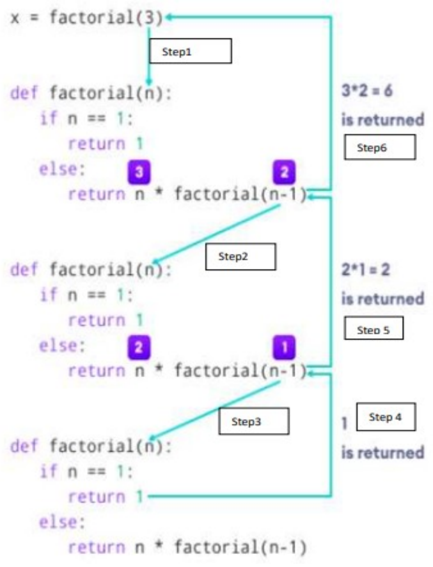
- They are inefficient as they take up a lot of memory and time.
- They can be computationally expensive.
- Hard to debug and sometimes the logic behind recursion is very difficult to understand.
- Recursion can lead to stack overflow errors if the recursion depth is too high.
- It can lead to infinite recursions if not used correctly.

Visualization of the program using python tutor is added below for your reference.





Flow Diagram of the program is added for your reference



```
x = factorial(3)

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

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

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

-END-