INTERMEDIATE PYTHON

LESSON 4 |**Recursion**| 17-12-18

***Video tutorial*:**

https://www.youtube.com/watch?v=06q7Gr4gPpA&index=4&list=PLqEbL1vopgvsv6jz8xB3Aa6w1njhf3egY

Python Recursion

Introduction

Recursion means iteration. A function is called recursive, if the body of function calls the function itself until the condition for recursion is true. Thus, a Python recursive function has a termination condition.

Why does a recursive function in Python has termination condition?

Well, the simple answer is to prevent the function from infinite recursion.

When a function body calls itself with any condition, this can go on forever resulting an infinite loop or recursion. This termination condition is also called **base condition**.

Is there any special syntax for recursive functions?

The answer is **NO**.

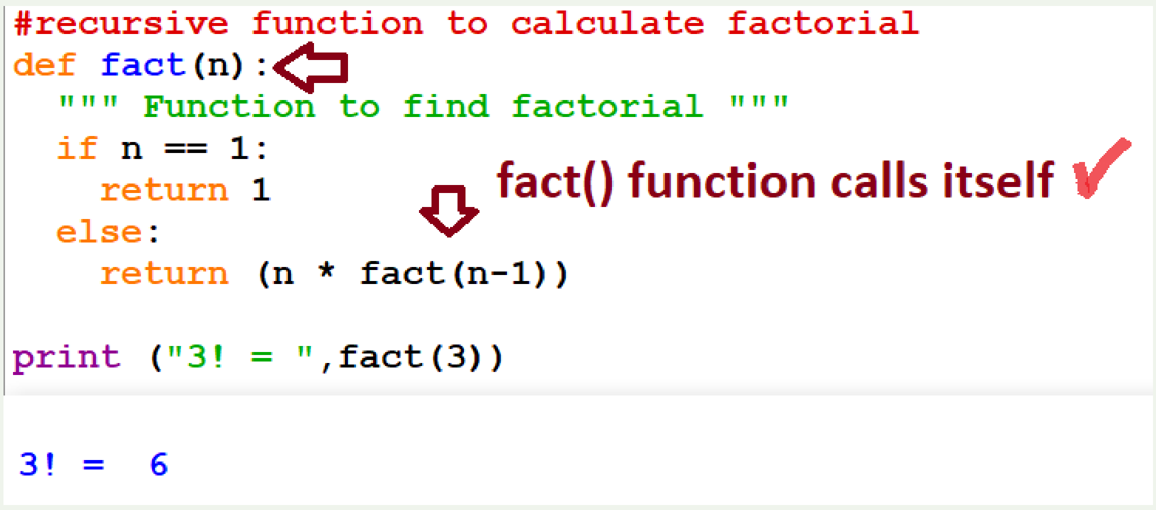
In Python, there is no syntactic difference between functions and recursive functions. There is only a logical difference.

Python Recursion: Example

Let's get an insight of Python recursion with an example to find the factorial of 3.

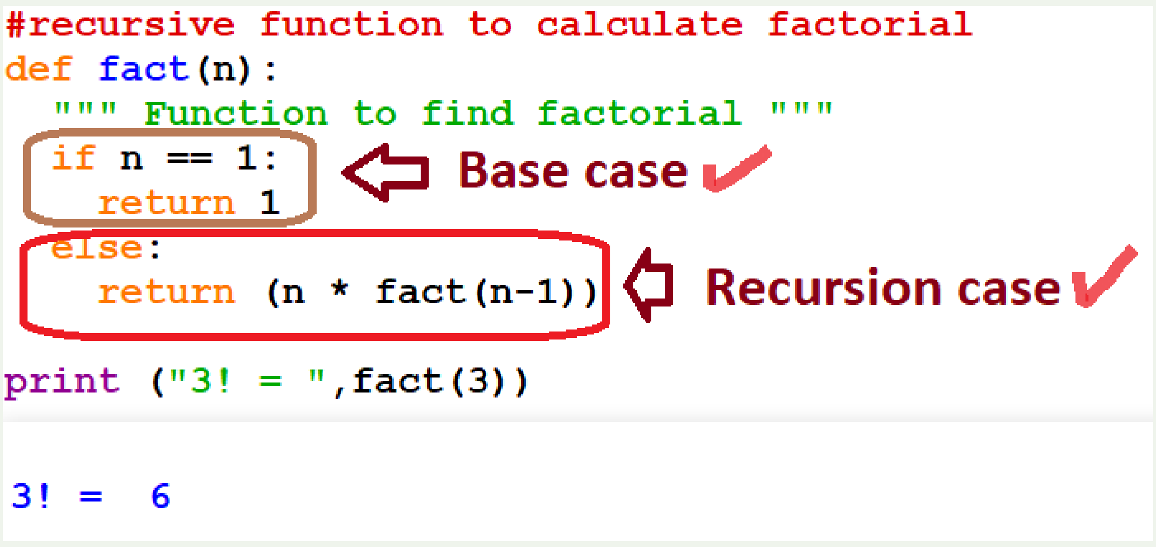
```
3! = 3 * 2! = 3 * (2 * 1!) = 3 * 2 * 1
```

This is how a factorial is calculated. Let's implement this same logic into a program.



```
#recursive function to calculate factorial
def fact(n):
  """ Function to find factorial """
  if n == 1:
    return 1
  else:
    return (n * fact(n-1))

print ("3! = ",fact(3))
```

```
3! =  6
```



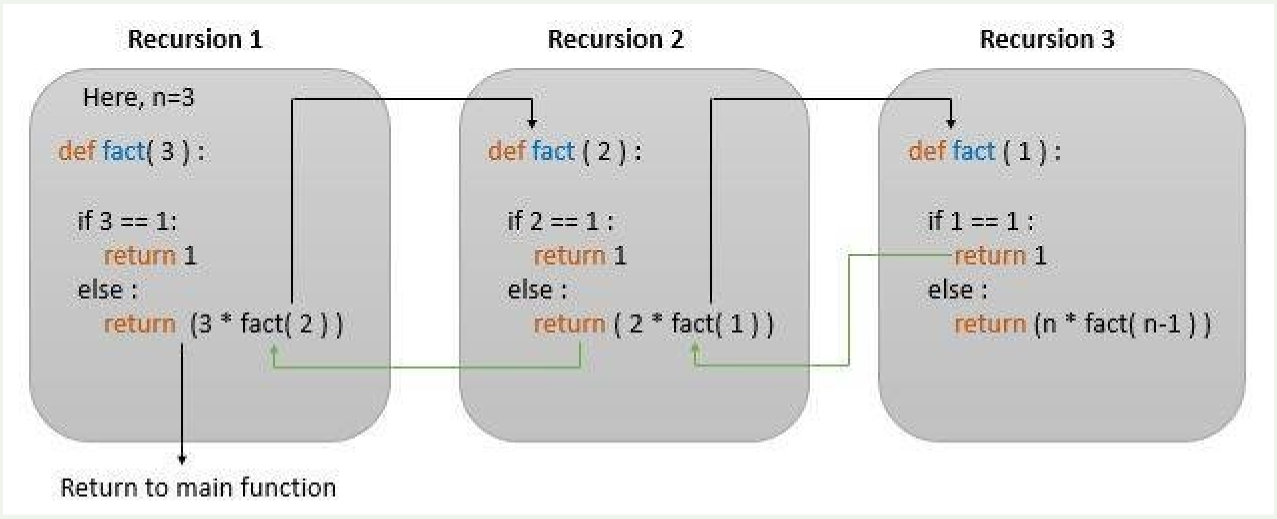
```
#recursive function to calculate factorial
def fact(n):
  """ Function to find factorial """
  if n == 1:
    return 1
  else:
    return (n * fact(n-1))

print ("3! = ",fact(3))
```

```
3! =  6
```

Explanation of the program

First is a base condition in any recursive function. If the value of n is equal to 1, then the function will return 1 and exit. Till this base condition is met, the function will be iterated.

In the program above, the value of the argument supplied is 3. Hence, the program operates in following way.



```
def fact( 3 ) :

  if 3 == 1:
    return 1
  else :
    return  (3 * fact( 2 ) )
```

```
def fact ( 2 ) :

  if 2 == 1 :
    return 1
  else :
    return ( 2 * fact( 1 ) )
```

```
def fact ( 1 ) :

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

When the function is called with the value of n

In recursion 1: Function returns 3 * fact(2). This invokes the function again with the value of n = 2.

In recursion 2: Function checks if n = 1. Since it's False function return 3 * 2 * fact(1). This again invokes the function with the value of n = 1

In recursion 3: Function checks if n = 1. This returns True making the function to exit and return 1.

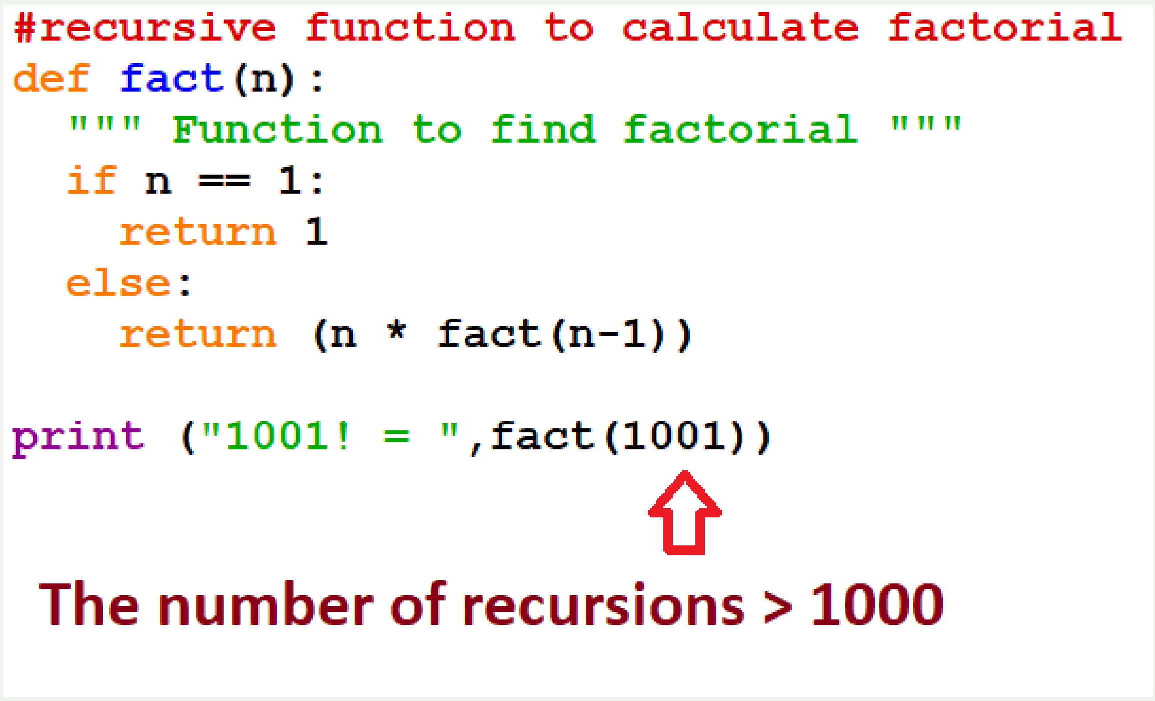
Hence after 3 recursions, the final value returned is 3 * 2 * 1 = 6.

Is there any limit on the number of recursions for a Python recursive function?

The answer is **YES**.

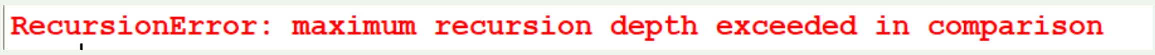
Unless we explicitly set the maximum limit of recursions, the program by default will throw a Recursion error after 1000 recursions.

Here is what happens when we supply 1001 as an argument in Python recursive function and the program have to call itself recursively over 1000 times.



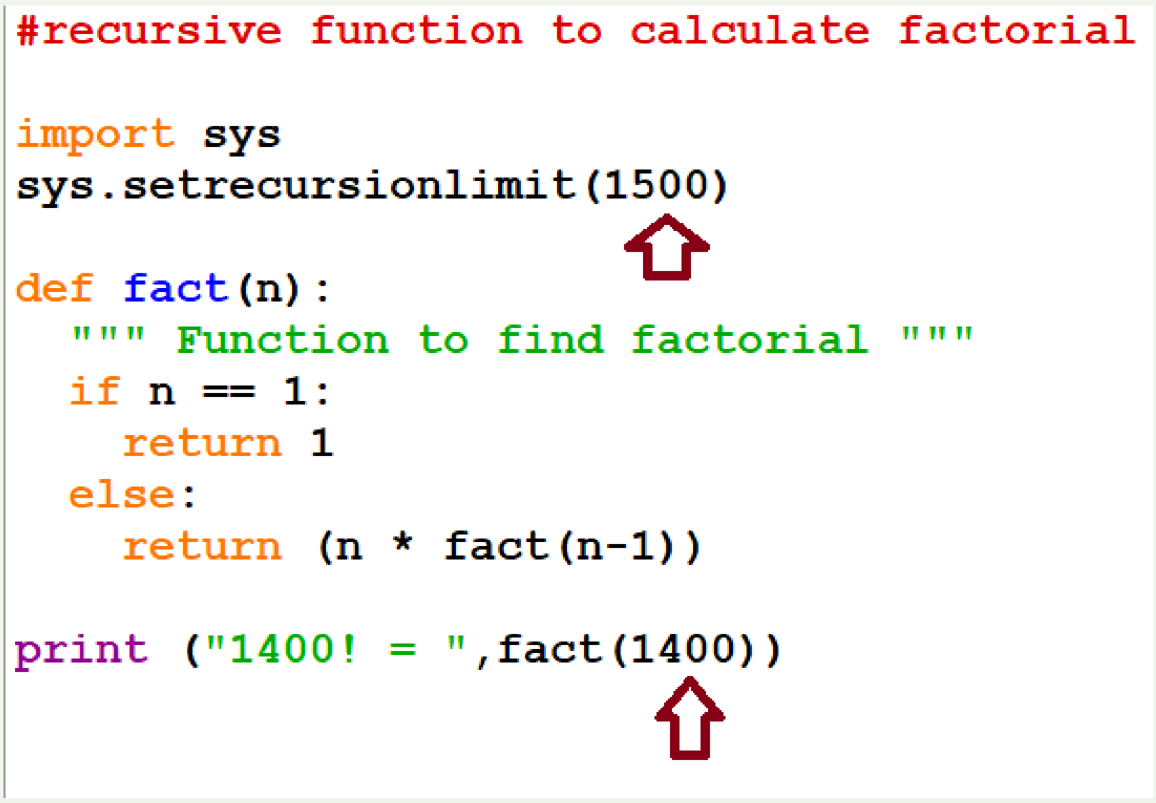
```python
#recursive function to calculate factorial
def fact(n):
  """ Function to find factorial """
  if n == 1:
    return 1
  else:
    return (n * fact(n-1))

print ("1001! = ",fact(1001))
```

Output:



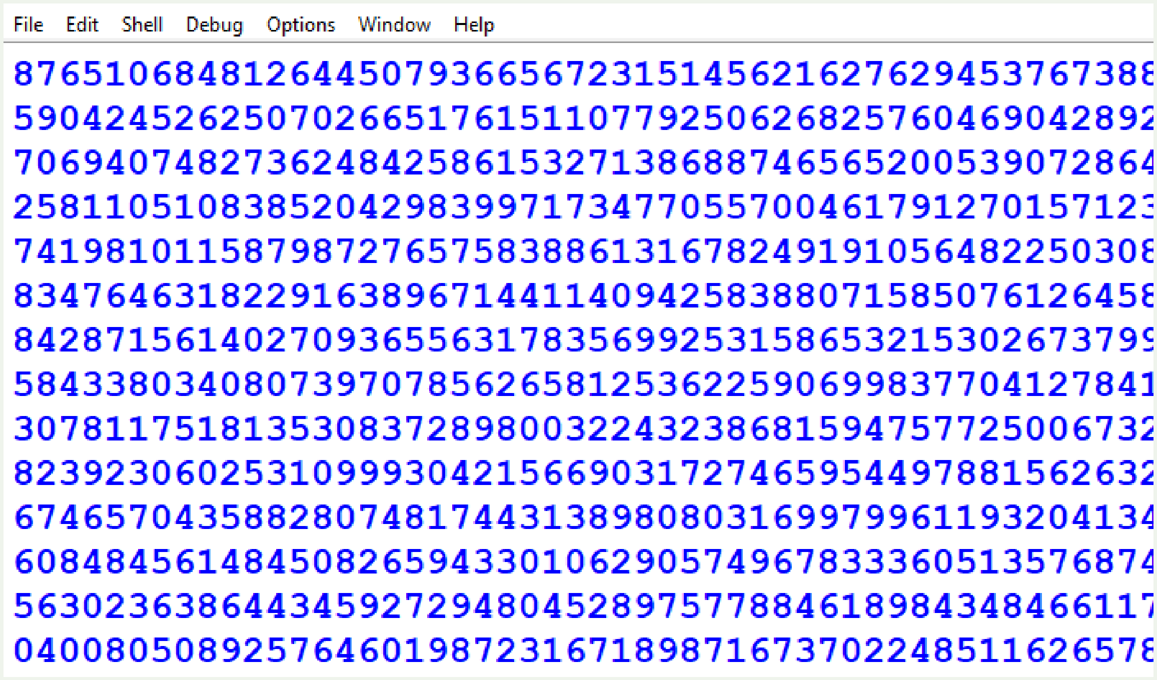
```
RecursionError: maximum recursion depth exceeded in comparison
```

To avoid such errors, we can explicitly set recursion limit using sys.setrecursionlimit( )



```
#recursive function to calculate factorial

import sys
sys.setrecursionlimit(1500)

def fact(n):
    """ Function to find factorial """
    if n == 1:
        return 1
    else:
        return (n * fact(n-1))

print ("1400! = ",fact(1400))
```

Output



```
87651068481264450793665672315145621627629453767388
59042452625070266517615110779250626825760469042892
70694074827362484258615327138688746565200539072864
25811051083852042983997173477055700461791270157123
74198101158798727657583886131678249191056482250308
83476463182291638967144114094258388071585076126458
84287156140270936556317835699253158653215302673799
58433803408073970785626581253622590699837704127841
30781175181353083728980032243238681594757725006732
82392306025310999304215669031727465954497881562632
67465704358828074817443138980803169979961193204134
60848456148450826594330106290574967833360513576874
56302363864434592729480452897577884618984348466117
04008050892576460198723167189871673702248511626578
```

Checkpoint

1. It is said that a recursive algorithm has more overhead thanan iterative algorithm. What does this mean?
2. What is a base case?
3. What is a recursive case?
4. What causes a recursive algorithm to stop calling itself?
5. What is direct recursion? What is indirect recursion?

Direct and Indirect Recursion

The example we have discussed so far show recursive function or function that directly call themselves. This is known as ***direct recursion***. There is also the possibility of creating ***indirect recursion*** in a program. This occurs when A function calls function B, which in turn calls function A. There can even be several functions involved in the recursion. For example, function A could call function B , which could call function C ,which calls function A .

Multiple Choice

1. A recursive function .

    a. calls a different function

    b. abnormally halts the program

    c. calls itself

    d. can only be called once

2. A function is called once from a program's function, then it

    itself four times. The depth of recursion is .

    a. one

    b. four

    c. five

    d. nine

3. The part of a problem that can be solved without recursion is thecase.

    a. base

    b. solvable

    c. known

d.terative

4. The part of a problem that is solved with recursion is the case.

   a. base

   b. iterative

   c. unknown

   d. recursion

5. When a function explicitly calls itself, it is called recursion.

   a. explicit

   b. modal

   c. direct

   d. indirect

6. When function A calls function B, which calls function A, it is called recursion.

   a. implicit

   b. modal

   c. direct

   d. indirect

7. Any problem that can be solved recursively can also be solvedwith a .

a. decision structure

b. loop

c. sequence structure

d. case structure

8. Actions taken by the computer when a function is called, such as allocating memory for parameters and local variables, are referred to as .

a. overhead

b. set up

c. clean up

d. synchronization

9. A recursive algorithm must in the recursive case.

a. solve the problem without recursion

b. reduce the problem to a smaller version of the original

problem

c. acknowledge that an error has occurred and abort the

program

d. enlarge the problem to a larger version of the original

problem

10. A recursive algorithm must in the base case.

a. solve the problem without recursion

b. reduce the problem to a smaller version of the original problem

c. acknowledge that an error has occurred and abort the program

d. enlarge the problem to a larger version of the original problem

True or False

1. An algorithm that uses a loop will usually run faster than an equivalent recursive algorithm.

2. Some problems can be solved through recursion only.

3. It is not necessary to have a base case in all recursive algorithms.

4. In the base case, a recursive method calls itself with a smaller version of the original problem.

Programming Exercises

1. Design a recursive function that accepts two arguments into the parameters x and y . The function should return the value of times

2. Write a recursive function that accepts an integer argument n,The function should display n lines of asterisks on the screen, with the the first line showing one asterisk, the second line showing two asterisks, up to the n th line which shows n asterisks.

3. Design a function that accepts a list as an argument and returns the largest value in the list. The function should use recursion to find the largest item.

4. Design a function that accepts an integer argument and returns the sum of all the integers from 1 up to the number passed as an argument. For example, if 50 is passed as an argument, the function will return the sum of 1, 2, 3, 4, . . . 50. Use recursion to calculate the sum.

5. Design a function that uses recursion to raise a number to a power. The function should accept two arguments: the number to be raised, and the exponent. Assume the exponent is anonnegative integer.

6. Write a program to print first 50 natural numbers using recursion.
7. Write a program to calculate the sum of numbers from 1 to n using recursion.
8. Write a program in C to Print Fibonacci Series using recursion
9. Write a program to print the array elements using recursion
10. Write a program to find the sum of digits of a number using recursion
11. Write a program to count the digits of a given number using recursion
12. Write a program to find GCD of two numbers using recursion
13. Write a program to get the largest element of an array using recursion
14. Write a program to reverse a string using recursion
15. Write a program to check a number is a prime number or not using recursion

16. Write a program to find the LCM of two numbers using recursion

17. Write a program to print even or odd numbers in given range using recursion

18. Write a program to calculate the power of any number using recursion

19. Write a program to find the first capital letter in a string using recursion

20. Design a recursive function that accepts an integer argument n, and prints the numbers 1 up through n .

21. So many occurrence in nature bear a recursive pattern, such as the famous nautilus shells, sunflowers, and various plants . Determine the base and the recursive case of the following functions: