Lecture-31

Anonymous functions:

In Python, anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

Syntax

The syntax of *lambda* functions contains only a single statement, which is as follows –

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Following is the example to show how *lambda* form of function works –

```
# Function definition is heresum = lambda arg1, arg2: arg1 + arg2;
# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

When the above code is executed, it produces the following result −

```
Value of total :  30
Value of
```

OR

```
sum = lambda arg1, arg2: arg1 + arg2;
```

OUTPUT:

```
>>> sum(34,32)
66
total :  40
```

```
 # Python code to illustrate cube of a number
# using labmda function
  cube = lambda x: x*x*x
print(cube(7))
```

Output:

```
343
```

Example2:

```
# using labmda function
cube =  lambda x: x**3+2*x+4
print(cube(3))
```

OUTPUT:

```
37
```

What is recursion?

Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Python Recursive Function:

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

Example of a recursive function

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

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

Output

```
>>> fact(3)
  6
```

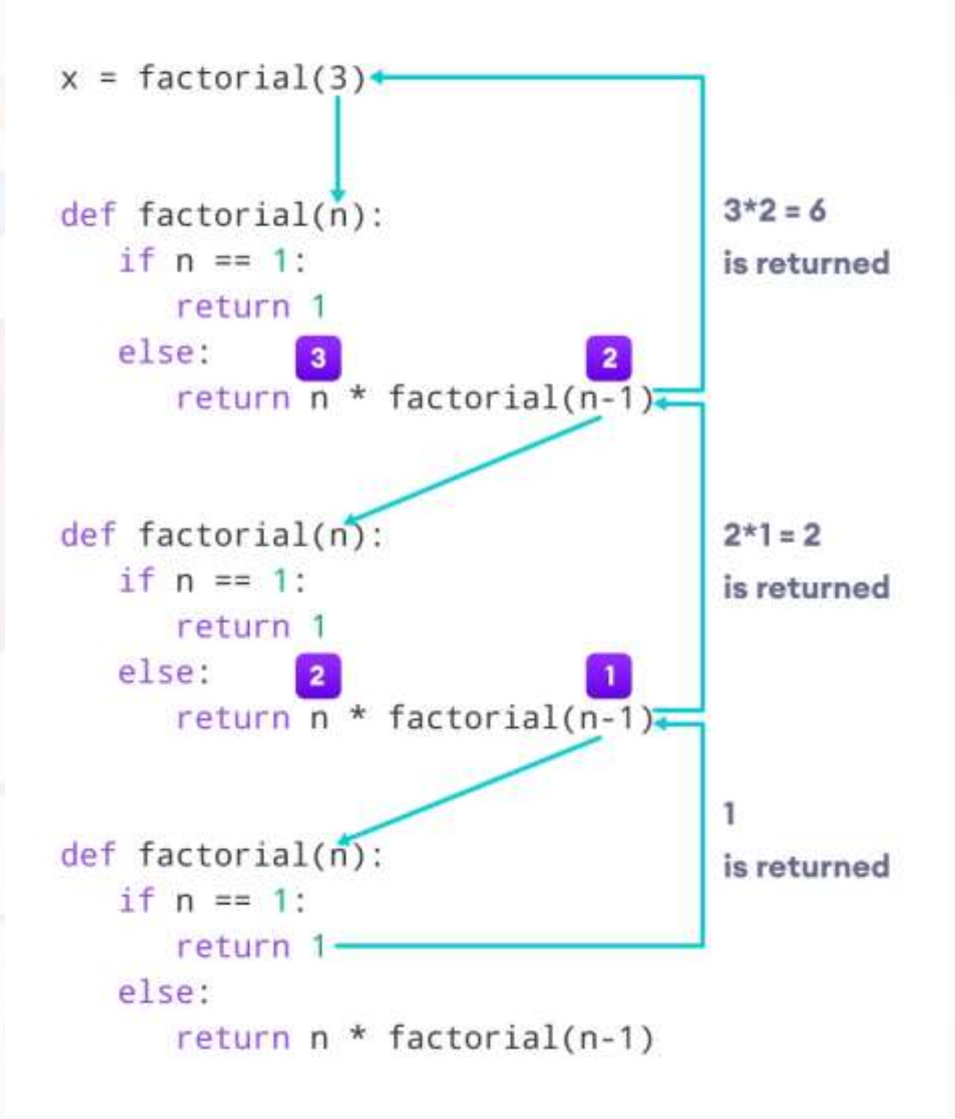
In the above example, factorial() is a recursive function as it calls itself.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

```
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
```

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



```
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)
```

Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$F_n = F_{n-1} + F_{n-2}$

with seed values

$F_0 = 0$ and $F_1 = 1$.

Method 1 (Use recursion) :

```
# Function for nth Fibonacci number
 def Fibonacci(n):
  if n<0:
    print("Incorrect input")
  # First Fibonacci number is 0
  elif n==1:
    return 0
  # Second Fibonacci number is 1
  elif n==2:
    return 1
  else:
    return Fibonacci(n-1)+Fibonacci(n-2)
 print(Fibonacci(9))
```

Output:

```
21
```

Method 2

```
# Program to display the Fibonacci sequence up to n-th term
 nterms = int(input("How many terms? "))
 # first two terms
n1, n2 = 0, 1
count = 0
 # check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
```

```
print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       # update values
       n1 = n2
       n2 = nth
       count += 1
```

Output

```
How many terms? 7
Fibonacci sequence:
0
1
1
2
3
5
8
```

References:

1.Introduction to Computation and Programming using Python, by John Guttag, PHI Publisher

2. Fundamentals of Python first Programmes by Kenneth A Lambert, Copyrighted material Course Technology Inc. 1 st edition (6th February 2009)

3. https://www.tutorialspoint.com/python/index.htm

4. https://www.geeksforgeeks.org/python-programming-language

5. https://www.w3schools.com/python/

****END OF THE LECTURE***

THANK YOU