→ "Python function within functions" :→

A function that is defined inside another function is known as the inner function or nested functions. Nested functions can access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

Eg:→

```
def f1():
    s = "welcome to Python Programming."

    def f2():
        print(s)

    f2()

f1()
```

O/P:- Welcome to Python Programming.

→ "Anonymous functions in Python" :→

In python, an anonymous function means that a function is without a name. We already know that "def keyword" is used to define a normal function. In order to create an anonymous function, "lambda keyword" is used.

Eg:→ Creating a normal function to return cube of a number.

```
def cube(x):
    return x*x*x
```

P.T.O

Creating an anonymous function to return cube of a number.

```
cube_1 = lambda x : x*x*x

print (cube(7))
print (cube_1(7))
```

O/p: 343
343

Eg2:→ anonymous function to find sum of first n natural numbers:

```
sum_natural = lambda n : (n*(n+1)) // 2
print (sum_natural(10))
print (sum_natural(5))
```

O/p: 55
15

Eg3:→ anonymous function to find out whether a number is prime or not.

```
is_prime = lambda n : n>1 and all(n%i!=0 for i in range
                                  (2, int(n**0.5)+1))

print (is_prime(7))
print (is_prime(10))
print (is_prime(2))
print (is_prime(1))
```

O/p: True
False
True
Fals

Explanation of this code :-

1. `n>1` → Ensures that the number is greater than 1 (since 1 is not prime).

2. `all(n%i!=0 for i in range(2, int(n**0.5)+1))`:

   Checks that n is not divisible by any number from 2 to $\sqrt{n}$.

   If n is divisible, it is not prime.

Eg:⇒ Create an anonymous function to extract first 10 prime numbers in a given range in the form of a list.

```
is_prime = lambda n: n>1 and all(n%i!=0 for i in
                        range(2, int(n**0.5)+1))
first_10_primes= list(filter(is_prime, range(2,100)))[:10]
print(first_10_primes)
```

Note:⇒ filter() function is used to extract prime numbers from range(2, 100):

& list(....)[:10]

Converts the filtered primes to a list.

Eg:⇒ Create an anonymous function to find out whether a number is even or odd.

P.T.O

```
even_odd = lambda n: "Even" if n%2 == 0 else "Odd"

print(even_odd(10))
print(even_odd(7))
print(even_odd(0))
print(even_odd(-3))
```

O/p:
Even
Odd
Even
Odd.

→ "Recursive functions in Python" :-

Recursion in Python refers to when a function calls itself.
Using a recursive function should be done with caution, as a recursive function can become like a non-terminating loop. It is better to check your exit statement while creating a recursive function

Eg:-

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

print(factorial(4))
```

O/p: 24.

PTO

Eg 2:→ Recursive program for fibonacci series :.

```
def fibonacci(n):
    return n if n<=1 else fibonacci(n-1) + fibonacci(n-2).
print(fibonacci(6))
```

O/p: 8.

```
nterms = 10
if nterms<=0:
  print("Please enter a the no")
else:
  print("Fibonacci sequence:")
  for i in range(nterms):
    print(fibonacci(i))
```

Eg 3:→ Sum of first n natural numbers :

```
def sum_natural(n):
    return n if n==1 else n+sum_natural(n-1)
print(sum_natural(5))
```

O/p: 15

Eg 4:→ Reversing a string using recursion :

```
def reverse_string(s):
    return s if len(s)==0 else s[-1] + reverse_string(s[:-1])
print(reverse_string("hello"))
```

O/p: olleh

or using slicing

```
def reverse_string(s):
    return s[::-1]
```

Note :→ s[-1] gets the last character of the string

s[:-1] Removes the last character & passes the remaining substring into the function.

Eg: 5 :> Checking whether a number is prime or not using recursion

```
def is_prime(n, i=2):
    if n <= 2:
        return True if n == 2 else False
    if n % i == 0:
        return False
    if i * i > n:
        return True
    return is_prime(n, i+1)
print(is_prime(29))
```

O/p. True.

Eg. 6 :> Finding GCD of two nos.

Let the two nos be 48 & 18

Step1 :> Find factors

Factors of 48 :- 1, 2, 3, 4, 6, 8, 12, 16, 24, 48.

Factors of 18 :- 1, 2, 3, 6, 9, 18

Step2 :> Find the greatest common factor

Common factors: 1, 2, 3, 6.

Largest one is 6 :. GCD(48, 18) = 6

Iterative approach to find GCD of two nos :-

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
print(gcd(48, 18))
```

O/p = 6.

Recursive approach to find GCD of two nos :-

```
def gcd(a, b):
    return a if b == 0 else gcd(b, a % b)
print(gcd(48, 18))
```

O/p = 6.

Eg7:- Recursive program for binary search :

```
def binary_search(arr, left, right, target):
    if left > right:
        return -1
    mid = (left + right) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] < targt:
        return binary_search(arr, mid+1, right, target)
    else:
        return binary_search(arr, left, mid-1, target)
```

```
arr = [1, 3, 5, 7, 9, 11]
print( binary_search(arr, 0, len(arr) - 1, 7))
```

→ Return statement in Python function:

The function return statement is used to exit from a function & go back to the function caller & return the specified value or data item to the caller. The syntax for the return statement is:

return [expression_list].

The return statement can consist of a variable, an expression, or a constant. which is returned at the end of the function execution. if none of the above is present. with the return statement, a None object is returned.

Example:-

```
def square_value(num):
    return num**2.

print(square_value(2))
print(square_value(-4))
```

O/p: 4
16.

→ Pass by Reference & Pass by Value:-

In python every variable name is a reference. When we pass a variable to a function, in python, a new reference to the

object is created.

Eg:>
```
def merafun(x):
    x[0] = 20
```
{ Here x is a new reference to the same list lst }

```
lst = [10, 11, 12, 13, 14, 15]
merafun(lst)
print(lst)
```

O/P: [20, 11, 12, 13, 14, 15]

Note:> When we pass a reference and change the received reference to something else, the connection between the passed and received parameters is broken.

Eg:>
```
def merafun(x):
    x = [20, 30, 40]
```
{ → Here the link of x with previous object gets broken and a new object is assigned to x.

```
lst = [10, 11, 12, 13, 14, 15]
merafun(lst)
print(lst)
```

O/P: [10, 11, 12, 13, 14, 15]

— o —