```
# Write once and use it as many time as you need
# Defining Function one time
def disp():
   name = "Pythonprogram"
   print("Welcome to", name)

# Calling Function as many time as we need
disp()
disp()
disp()

#Divide Large task into many small task, helpful
for debuging code
# Seprate Function for Addition
def add():
   x = 10
   y = 20
   c = x + y
   print(c)

add()

# Seprate Function for Subtraction
def sub():
   x = 10
   y = 20
   c = y - x
   print(c)

sub()
```

****

```
# Function without Argument and Parameter
```

```python
#Defining a Function without Parameter
def add():
   x = 10
   y = 20
   c = x + y
   print(c)

#Calling a Function without Argument
add()

#Defining a Function with Parameter
def add(y):
   x = 10
   c = x + y
   print(c)

#Calling a Function with Argument
add(20)

#Defining a Function with Parameter
def add(y):
   x = 10.2334
   print(x+y)
   print(f"Formatted Output {x+y:5.2f}")

#Calling a Function with Argument
add(20)
```

****

```python
# Return Statement Single Value
#Defining a Function
def add():
   x = 10
   y = 20
   c = x + y
   return c
```

```
#Calling a Function
sum = add()
print(sum)
# print(sum())
print()


#Defining a Function
def add():
   x = 10
   y = 20
   return x + y

#Calling a Function
sum = add()
print(sum)

#Defining a Function with Parameter
def add(y):
   x = 10
   return (x + y)

#Calling a Function with Argument
sum = add(20)
print(sum)

# Return Statement Multiple Values
#Defining a Function
print("Return Statement Multiple Values")
def add(y):
   x = 100
   c = x + y
   d = y - x
   c1 = x - 50
   return c, c1, d

#Calling a Function
```

```
sum, sub, a,b = add(20)
print(sum)
print(sub)
print(a)
```

****

```
#Nested Function
# Example 1
def disp1():
   def show1():
      print("Show Function")
   print("Disp Function")
   show1()

disp1()

# Example 2 With Return Statement
def disp2():
   def show2():
      return "Show Function "
   result = show2() + "Disp Function"
   return result
print(disp2())

# Example 3 With Return Statement and Parameter
def disp(st):
   def show():
      return "Show Function "
   result = show() + st + " Disp Function"
   return result
print(disp("Welcome"))
```

****

```
#Pass a Function as Parameter
# Example 1
def disp(sh):
    print(type(sh))
    print("Disp Function" + sh())

def show():
    return " Show Function"

disp(show)

# Example 2 with return
def disp(sh):
    return "Disp Function" + sh()

def show():
    return " Show Function"

result = disp(show)
print(result)
```

****

```
#Function Return another Function
# Example 1
def disp():
    def show():
        return "Show Function"
    print("Disp Function")
    return show

r_sh = disp()
print(r_sh())

# Example 2
def disp(sh):
    print("Disp Function")
    return sh

def show():
    return "Show Function"
```

```
r_sh = disp(show)
print(r_sh())
```

****

```
#Positional Arguments
#Example 1
def pw(x, y):
   z = x**y
   print(z)

pw(5, 2)

#Example 2
def pw(x, y):
   z = x**y
   print(z)

pw(2, 5)

#Example 3 will show Error
#def pw(x, y):
#   z = x**y
#   print(z)

#pw(5, 2, 3)
```

****

```
#Keyword Arguments
#Example 1
def show(name, age):
   print(f"Name: {name} Age: {age}")

show(name="Ram", age=62)

#Example 2
```

```
def show(name, age):
    print(f"Name: {name} Age: {age}")

show(age=62, name="Shyam")

#Example 3 will show Error
#def show(name, age):
#   print(f"Name: {name} Age: {age}")

#show(name="GeekyShows", age=62, roll=101)
```

```
*****************************************************************************
```

```
#Default Arguments
#Example 1
def show(name, age):
    print(f"Name: {name} Age: {age}")

show(name="Ramesh", age=62)

#Example 2
def show(name, age=27):
    print(f"Name: {name} Age: {age}")

show(name="Ramesh")

#Example 3
def show(name, age=27):
    print(f"Name: {name} Age: {age}")

show(name="Ramesh", age=62)

#Example 4 will show Error
#def show(name, age=27):
#   print(f"Name: {name} Age: {age}")

#show(name="GeekyShows", age=62, roll=101)
```

```
******************************************************************************

#Variable Length Arguments
#Example 1
def add(x, y):
   z = x+y
   print("Addition:", z)

add(5, 2)

#Example 2
def add(*num):
   z = num[0]+num[1]+num[2]+num[3]
   print("Addition:", z)

add(5, 2, 4, 9)

#Example 3
def add(x, *num):
   z = x+num[0]+num[1]
   print("Addition:", z)

add(5, 22, 4)

******************************************************************************

#Keyword Variable Length Arguments
#Example 1
def add(**num):
   z = num['a']+num['b']+num['c']
   print("Addition:", z)

add(a=5, b=2, c=4)

#Example 2
def add(x, **num):
   z = x+num['a']+num['b']
   print("Addition:", z)
```

```
add(3, a=5, b=2)

*******************************************************************************

# Local Variable
# Example 1
def show():
   x = 10    # Local Variable
   print(x)   # Accessing Local Variable inside
Function
show()
#Accessing Local Variable outside Function
# print(x)    # It will show error

# Example 2
def add(y):
   x = 10    # Local Variable
   print(x)   # Accessing Local Variable inside
Function
   print(x+y) # Accessing Local Variable inside
Function

add(20)
#Accessing Local Variable outside Function
# print(x)    # It will show error

*******************************************************************************

# Global Variable
# Example 1
a = 50
def show():
   a = 10    # Local Variable
   print(x)   # Accessing Local Variable inside
Function
   print(a)   # Accessing Global Variable inside
Function
```

```
show()
# Accessing Global Variable outside Function
print("Global Variable A:",a)

# Accessing Local Variable outside Function, show
error
#print("Global Variable X:",x)

# Example 2
i = 10
def myfun():
   a1 = i + 1
   print("My Function", a1)

myfun()

# Example 3
i = 0
def myfun():
   # We are trying to increase global variable
   # but remember here i is treated as local
variable with same name
   # and as we dont referenced it show it will show
error
   i = i + 1
   print("My Function", a)

myfun()
```

****

```
# Global Keyword
#Example 1
a = 50
def show():
   a = 10
   print(" E1 - A:",a)        # It will show local
variable value
```

```
show()
print("A:",a)           # It will show global variable
value

#Example 2
a = 50
def show():
   global a
   print("E2 - A:",a)
   a = 20           # Modifiying Global Variable
value
   print("E2 - A:",a)
show()
print("E2- outside A:",a)        # It will show
modified global variable value
```

```
**************************************************************************
```

```
# globals ( ) Function
a = 50
def show():
   a = 10
   print("Local Variable A:",a)
   x = globals()['a']
   print("X:",x)
   x = 40
   print("X:",x)
show()
print("Global Variable A:",a)
#print("X:",x)
```

```
**************************************************************************
```

```
# Pass/Call by Object Reference
def val(x):
   x = x + 25
   print(x, id(x))
```

```
x = 10
val(x)
print(x, id(x))
```

```
********************************************************************************

# Pass/Call by Object Reference
def val(lst):
   print("Inside Function Before Append:", lst,
id(lst))
   lst.append(4)
   print("Inside Function After Append:", lst,
id(lst))

lst = [1, 2, 3]
print("Before Calling Func:", lst, id(lst))
val(lst)
print("After Calling Func:", lst, id(lst))
```

```
********************************************************************************

# Pass/Call by Object Reference
# Example 3
def val(lst):
   print("Inside Function Before New:", lst,
id(lst))
   # Create New list object
   lst = [11, 22, 33]
   print("Inside Function After New:", lst,
id(lst))

lst = [1, 2, 3]
print("Before Calling Func:", lst, id(lst))
val(lst)
print("After Calling Func:", lst, id(lst))
```

```
********************************************************************************
# Recursion
********************************************************************************

# Example 1
i = 0
def myfun():
   global i
   i+=1
   print("My Function", i)
   myfun()

myfun()

# Example 2
# import sys
# get recursion limit
# print("Default:",sys.getrecursionlimit())

# set recursion limit
# sys.setrecursionlimit(3000)

# print("After setting:", sys.getrecursionlimit())

********************************************************************************

# Factorial using Recursion
def fact(n):
   if n == 0:
      return 1
   return n * fact(n-1)

print(fact(5))



# Python program to display the Fibonacci sequence
```

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

nterms = 10

# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
       print(recur_fibo(i))
```

Example – 2 Fib function

```
# Function for nth Fibonacci number
def Fibonacci(n):

    # Check if input is 0 then it will
    # print incorrect input
    if n < 0:
        print("Incorrect input")

    # Check if n is 0
    # then it will return 0
    elif n == 0:
        return 0

    # Check if n is 1,2
    # it will return 1
    elif n == 1 or n == 2:
        return 1

    else:
        return Fibonacci(n-1) + Fibonacci(n-2)

# Driver Program
print(Fibonacci(9))
```

Example - 3

```
# Function for nth fibonacci
# number - Dynamic Programing

# Taking 1st two fibonacci nubers as 0 and 1
FibArray = [0, 1]

def fibonacci(n):
    # Check is n is less
    # than 0
    if n <= 0:
        print("Incorrect input")

    # Check is n is less
    # than len(FibArray)
    elif n <= len(FibArray):
        return FibArray[n - 1]
    else:
        temp_fib = fibonacci(n - 1) +
                   fibonacci(n - 2)
        FibArray.append(temp_fib)
        return temp_fib

# Driver Program
print(fibonacci(9))
```

Example – 4

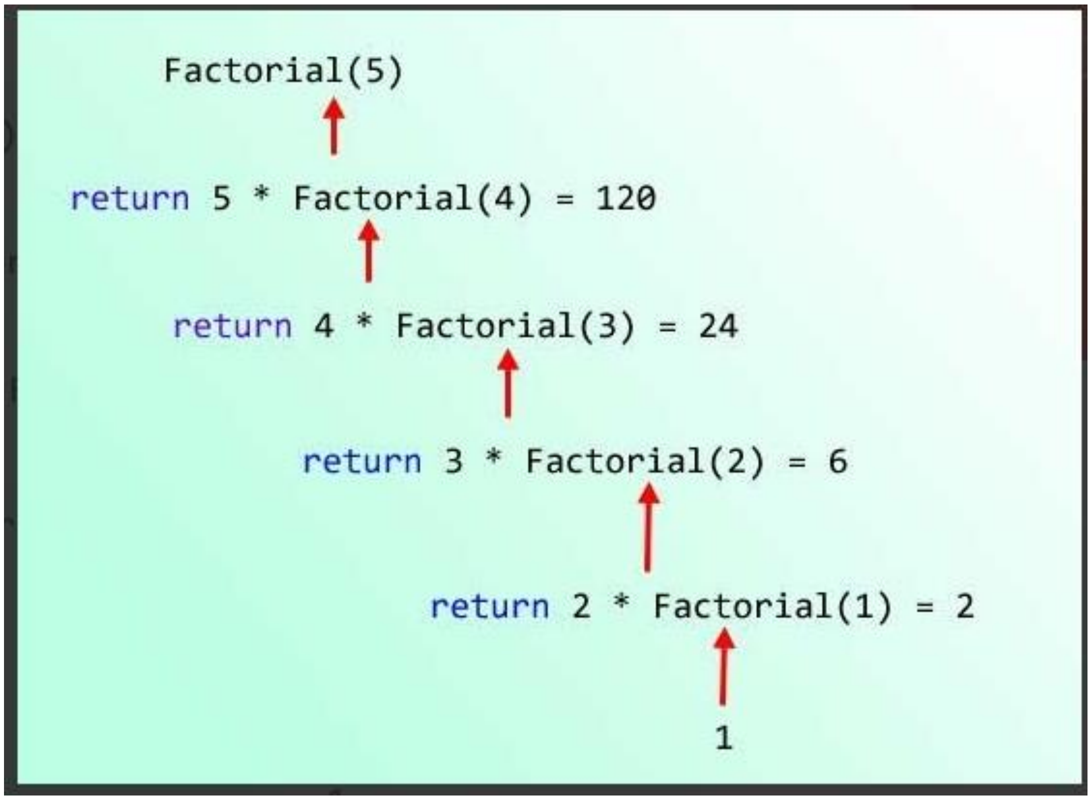
```
# Function for nth fibonacci
# number - Space Optimisataion
# Taking 1st two fibonacci numbers as 0 and 1
```

```
def fibonacci(n):
    a = 0
    b = 1

    # Check is n is less
    # than 0
    if n < 0:
        print("Incorrect input")

    # Check is n is equal
    # to 0
    elif n == 0:
        return 0

    # Check if n is equal to 1
    elif n == 1:
        return b
    else:
        for i in range(1, n):
            c = a + b
            a = b
            b = c
        return b
```

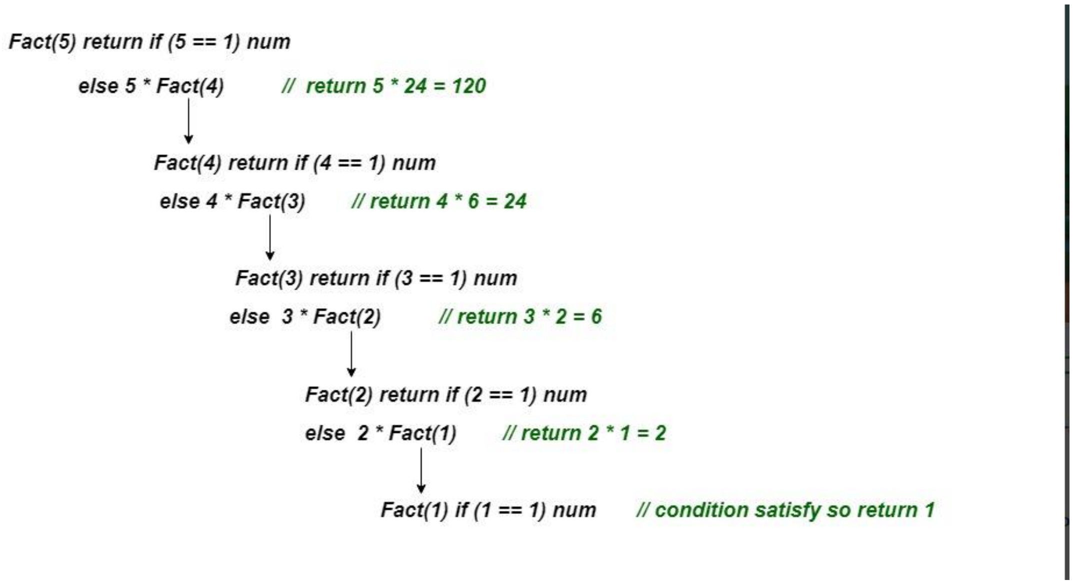
```
# Driver Program
print(fibonacci(9))
```

Fib using loop

```
# Enter number of terms needed
#0,1,1,2,3,5....
a=int(input("Enter the terms"))
f=0                                                   #first
element of series
s=1                                                   #second
element of series
if a<=0:
    print("The requested series is",f)
else:
    print(f,s,end=" ")
    for x in range(2,a):
        next=f+s
        print(next,end=" ")
        f=s
        s=next
```



```
     Factorial(5)
          ↑
return 5 * Factorial(4) = 120
              ↑
     return 4 * Factorial(3) = 24
                   ↑
          return 3 * Factorial(2) = 6
                        ↑
               return 2 * Factorial(1) = 2
                             ↑
                             1
```



```
Fact(5) return if (5 == 1) num
    else 5 * Fact(4)        //  return 5 * 24 = 120
           ↓
        Fact(4) return if (4 == 1) num
         else 4 * Fact(3)        // return 4 * 6 = 24
                ↓
            Fact(3) return if (3 == 1) num
            else  3 * Fact(2)        // return 3 * 2 = 6
                    ↓
                Fact(2) return if (2 == 1) num
                else  2 * Fact(1)        // return 2 * 1 = 2
                        ↓
                    Fact(1) if (1 == 1) num      // condition satisfy so return 1
```

****

Anonymous Function or Lambda Function

****

```
#Example 1 Single Argument
show = lambda x : print(x+2)
show(5)

#Example 2 Two Arguments
add = lambda x,y : (x+y)
print("2....",add(5, 5))

#Example 3 Return Multiple
add_sub = lambda x,y : (x+y, x-y)
a, s = add_sub(5, 2)
print("This is a = ",a)
print("This is s = ",s)

#Example 2 with Default Argument
add = lambda x,y=3 : (x+y)
print(add(5,8))
```

****

```
# Nested Lambda Function
add = lambda x=10 : (lambda y : x + y)
a = add()
# print(a)
print(a(20))


#****************
add = lambda x : (lambda y : x + y)
a = add(100)
```

```
# print(a)
print(a(20))

*****************************************************************************

# Passing Lambda Function to Another Function
def show(a):
   print(a)
   print(a(8))

show(lambda x: x)
*****************************************************************************

# Return Lambda Function
def add():
   y = 20
   return (lambda x : x+y)

a =add()
print(a(10))

*****************************************************************************


*****************************************************************************

(lambda x : print(x + 7))(5)
(lambda x, y : print(x + y))(55, 22)

*****************************************************************************
```

# Function Decorator

```
*****************************************************************************

# # Example 1
# def decor(num):
#  def inner():
#     print("Inner Function: Before enhancing
Function")
#     num()
#     print("Inner Function: After enhancing
Function")
#   return inner
#
# @decor
# def num():
#   print("We will use this function")
#   print("and will enhance this in decorator")
#
#
# # num = decor(num)
# num()

# Example 2
def fun1(num):
   def inner():
      num()
      print("Inner Function: Before enhancing
Function")
      print("Inner Function: After enhancing
Function")
   return inner

def fun2(fun):
   def inner():
      print("line 1")
      print("Line 2")
      fun()
      print("Line 5")
   return inner

def fun3(fun):
   def inner():
      print("line 1")
      print("Line 2")
      fun()
      print("Line 5")
      print("Line 6")
```

```
        print("Line 7")
    return inner


def num():
    print("num line 1")
    print("Num line 2")

# result_fun = decor(num)
# result_fun() #instead this directcly call num
function
num()
```

```
*****************************************************************************
```

```
# Example 1
# def decor(fun):
#   def inner():
#       a = fun()
#       add = a + 5
#       return add
#   return inner
#
# def num():
#   return 10
#
# result_fun = decor(num)
# print(result_fun())

# Example 2
def decor(fun):
    def inner():
        a = fun()
        add = a + 5
        return add
    return inner
```

```
@decor
def num():
   return 10
#
# #result_fun = decor(num)
# #print(result_fun()) instead this directcly call
num function
print(num())
```

```
******************************************************************************
```

```
# Two Decorator Function to same function
# Example 1
# def decor(fun):
#  def inner():
#     a = fun()
#     add = a + 5
#     return add
#  return inner
#
# def decor1(fun):
#  def inner():
#     b = fun()
#     multi = b * 5
#     return multi
#  return inner
#
# def num():
#  return 20
#
# result_fun = decor(decor1(num))
# print(result_fun())

# Example 2
def decor(fun):
   def inner():
      a = fun()
```

```
        add = a + 5 #55
        return add
    return inner

def decor1(fun):
    def inner():
        b = fun()
        multi = b * 5  # 50
        return multi
    return inner
# @decor
@decor1
def num():
    return 10

# result_fun = decor(decor1(num))
# print(result_fun()) #instead of this directly
call num function
print(num())
```

****

Generator

Generators are functions that return a sequence of values. We use yield statement to return the value from function.

Yield Statement

Yield statement returns the elements from a generator function into a generator object.

Ex:- yield a

next ( ) Function

This function is used to retrieve element by element from a generator object.

Syntax:- next(gen_obj)

****

```
# # Example 1
def disp(a,b):
   yield a
   yield b

x,y = disp(10, 20)
print(x)
print(y)
print()

# # Example 2
def disp(a,b):
   yield a
   yield b
result = disp(10, 20)
print(result)
print(type(result))
# converting to list
lst = list(result)
print(lst)
print(type(lst))
```

****

```
def disp(a,b):
   yield a
   yield b

result = disp(10, 20)

print(result)
print(type(result))

print(next(result))
print(next(result))
```

****

```
def show(a,b):
   while a<=b :
      yield a
      a+=1
result = show(1, 5)
# print(result)
print(type(result))

print(next(result))
# print(next(result))
# print(next(result))
# print(next(result))
# print(next(result))
for i in result:
   print(i)
```

****