function :-

1) user defined

2) Built in funcn

Ex:- Print ( ), Sum ( ), type ( )
len ( )

type ( ) :- This funcn return types of Object or data type of variable

Ex:-

```
list_fruits = ('apple', 'banana', 'chos
                mango')
print (type (list_of_fruits))
```

O/P

<class 'tuple'>

abs ( ) :- This funcn return absolute valu of the specified number.

Ex:-

```
negative_number = -676
print ( abs (negative_number))
```

O/P

676 (It removes '-' sign cha it into +ve).

pow ( ) → It return the calculated valu of x to the power of y. i.e; x

Ex :-

```
x = pow (3, 4)
print (x)
```

O/P

&

round() → It returns the nearest integer of a specified floating point number

Ex: round(88.764)
o/p
88

→ user defined funcn:— The funcn which we define ourself in a program is known as user defined funcn.

→ Keywords are not used as the name of funcn.

→ In python we defind UDF by using keyword 'def' followed by the funcn name.

Syn:—
```
def name_of_funcn():
    Statement
```

UDF funcn are of 4 types:—
I) funcn without return and without parameter
II) " " " with "
III) " with " " out "
IV) " " " "

1) funct without return and without parameter

Ex:—

```
def Sum():
    print("enter two number")
    a = float(input())
    b = ""  ""
    s = a+b
print("Sum is:", s)
```

2) funct with parameter and without return

Ex:

```
def Sum(a,b):
    s = a+b
    print("Sum is:", s)

print("enter two number:")
a = int(input())
b = ""  ""
Sum(a,b)
```

3) funct without parameter and with return

Ex

```
def Sum():
    print("Enter two number")
    a = int(input())
    b = ""
    s = a+b
    return s

c = Sum()
print("Sum is", c)
```

iv) funcls with parameter and with return.

```
def Sum(a, b):
    s = a+b
    return s

print ("enter two number")
a = int (input ())
b = int ("  ")
c = Sum(a, b)
print ("Sum is", c)
```

Scope :-
The variable is only available from inside the region it is created This is called scope or local variable.

Ex:-

```
def my_func():
    x = 300
    print(x)

my_func().
```

→ Function inside another function - Local scope:

```
def myfunc():
    x = 300
    def myinnerfunc()
        print(x)
    my_innerfunc()

myfunc()
```

Global scope :-
The variable created in the main body of the python code is called global variable and belongs to the global scope.

Global variable scope is both global and local.

Or variable created outside the funch is used in funch or outside also

Ex:-

```
x = 300

def myfunc():
    print(x)

myfunc()
print(x)
```

→ **Global Keyword:-** If we want to use a local variable of a funct as a global variable then we have to use 'global' keyword.

Syn:- global variable name.

Ex:-

```
def my-func():
    global x
    x = 300

my func()
print(x)
```

→ **Lambda funct:-** A lambda funct is uses to write funct in a single line

Syn:-

lambda arguments : expression

Ex:- lambda x=400 : print(x)

In lambda funct their are may be more than 1 argument but only one expression.

Ex :-

Add 10 to argument a, and return result.

lambda funct<u>n</u> argument

y = lambda <u>a</u> : <u>a+10</u> → expression

print (y(5))

Ex :-

print ("Enter value of a & b")

a = int(input( ))

b = int(input( ))

x = lambda a, b : a * b

print (x(a, b))

→ Lambda funct<u>n</u> is also called anonymous funct<u>n</u>. It is also used inside other funct<u>n</u> or it is used as a inner funct<u>n</u>.

→ If their is funct<u>n</u> that takes one argument and that argument will be multiplied with an unknown number

def my-func(n):

return lambda a : a * n

Ex :-

def myfunc (n):

return lambda a : a*n

my-doubler = myfunc (2)

print (my-doubler(11))

Or:-

```
def my-func(n):
    return lambda a: a*n

print("enter the value of n")
n = int(input())
print("enter the value of a")
a = int(input())
my_result = my-func(n)
print(my_result(a))
```

→ we have to pass lambda argument value

→ **Recursion or Recursive function :-**

When funcn call itself than this is called recursion or recursive funcn.

```
def recurse():
    ___ ___
    recurse1()
    ___
    recurse()
```

recursive call

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$5! = 5 \times (4!)$

$= 5 \times 4 \times (3!)$

Ex:-

funcn to find factorial of entered number

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return (num * factorial(num-1))
print("enter number")
```

```
num = int (input ())
print ("factorial of", num, "is");
factorial (num)
```

x:—
```
def myfun (x, y=50):
    print ("x:", x)
    print ("y:", y)

myfun (10)

myfun (10,20)
```

→ o/p : x: 10
y: 50 → It print the value which is pass

→ o/p : x: 10
y: 20 → It will change the previously assign value with new passed value.

Arbitrary Arguments, *args :— By using this we can pass variable number of argument in a funch we not need to define the number of arguments passed.

Ex :—
```
def mykids (*kids):
    print ("the youngest child is", kids[2])

mykids ("Emil", "Abasis", "Linus")
```
→ Variable name of list created

o/p :-
The youngest child is Linus

Page:
Date: 6/3/24

Reverse of string without using reverse funct

```
print ("enter string :")
str1 = (input())
len1 = len(str1)
for j in range(-1, -(len(str1)+1), -1):
                    (len1+1)
    Print(str[j], end = "")
```

> If we use this than not need to find length of string.

**kwargs, Arbitrary keyword arguments :- This way of argument will receive a dictionary of arguments that is it store data with key and its attributes or value.

Syn :- def func-name(** argument-name) (variable)

Ex :- def myfun2(**kwargs):

```
    print("his father name is", kwargs["fname"])

myfun2(fname = 'abc', iname = 'S')
```

Decorator ~~function~~ :— Python decorators are a powerful tool that allow to mo the behaviour of previously defined funcn() or method().

A decorator is a funcn that takes another funcn as an argument and returns a new modifies behaviour of the original function. The new fun is called 'decorated' funcn.

Syn:—

```
        (new)
@decorator_funcn name
```

Ex:-

```
@ great ( )
    def oldfuncn_name():
```

```
@ add
    def name ():
        print ("my name is niharika")

    name ()

    def add (x):
        def add1 ():
            print ("I live in Kolkat
            x()
            print ("Thankyou")
        return add1
```

O/P

```
I live in SaltLake Kolkata
My name is niharika
Thankyou
```

To use "*args" and "**kwargs" in Decorator :—

```
@greet1
def sub(a,b):
    print (a-b)

sub(4,2)

def greet1(x):
    def greet2(*args,**args):
        print("goodmorning")
        x(*args,**args)
        print("thankyou")
    return greet2
```