**FUNCTION:**

It is one building block, which contains variables and operations. This single unit is very useful to execute the code inside it, multiple times.

In Python, a function is a group of related statements that performs a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes the code reusable.

**SYNTAX:**

def function\_name(parameters):

variables

operations

**TWO IMP APPROACHES RELATED TO A FUNCTION**

**1. Declaration of a function**:

creating/writing a complete definition of it.it provide the structure of a function.

Example: Write a function to add 2 numbers

def add(): #declaration

print(100+400)

Only declaration doesnt give u an output to get an output u need to call the function

**2. Calling of a function means: we actually allocate to the memory**

**syntax:**

function\_name()

**Function Declaration and calling**

def add(): #declaration

print(100+400)

# calling of add

add()

**How many times, we can call a function??**

Ans: n times

We pass arguments in a function, we can pass no arguments at all, single arguments or multiple arguments to a function and can call the function **multiple times**.

**Interview question: Advantage of a function** ==>

1. Code Re-usability (With the help of functions, we can avoid rewriting the same logic or code again and again in a program)
2. Declaration is only ones, but calling is multiple times, as per the need.
3. We can track a large Python program easily when it is divided into multiple functions.

Example:

def add ():

print(10+20)

a = 3

b = 4

add ()

from math import pi

print(pi)

add ()

add ()

-------------------------

def add(a,b):

print(a+b)

a = 3

b = 4

add(a,b)

from math import pi

print(pi)

add(10,40)

add(300,400)

--------------------------------

**We have2 types of function**

**1. Built in function:** present in python itself

Example:

print(), type(), id(), input() etc

**2. User defined function:** created by a user as per business requirement

Example:

def add(a,b):

print(a+b)

------------------

def add():# declaration

a = 200 # var a

b = 100 #var b

print(a+b) #print func

add() # calling

##############################

**Types of variables**

1. **Local Variables:**

Whenever we declairing the variables these are available only inside the scope/block

outside the block, local var's will not be available.

def add ():# declaration

a = 200 # var a

b = 100 #var b

print(a+b) #print func

add () # calling

print (a +b) # a, b local var's not accessible outside

-----------------------------

If a,b i want to access them inside a function and outside it then???

MAKE it global

**2. Global Variable:** which is accessible everywhere, when we declare the variables outside and before the declaration of a function

--------------

# global a, b

a = 20

b = 40

def add():

print('Inside:', a+b)

add()

print ('Outside:', a+b)

x = 100

def sample():

x = 10

print(x)

sample()

print(x+200)

=======================

x = 100

def sample():

#print(x+100)

#let’s try to change value of a global x

x = x + 100

print(x)

sample()

# This will give u Unbound Local error

means we are not allowed to change global value inside a local scope

But if u have a local x then definitely u can change value of it

x = 100

def sample():

#print(x+100)

x = 20 # local

# lets try to change value of a global x

x = x + 100

print(x)

sample()

===================

But if we want to change global x value then??????

Ans : use global keyword

example: global var

x = 100

def sample():

global x # it will allows u to change global value inside a local scope

# lets try to change value of a global x

x = x + 100

print(x)

sample()

print(x)

**Another approach of a function:**

Syntax:

def func(paramter/s):

return

func()

------------------

**return**: when we want to return something to the function calling

then will use a return statement/keyword

If we want something to fetch outside the function from inside the function then use return

# when no return

def sample():

pass

print(sample())

# It returns None when no return

------------------------

# when return is given

def sample():

return 'Pooja'

print(sample())

# It returns specified things along with calling

==============================

General example:

print(print(45))

# print doesnt have return==> None

print(id(45))

# id returns an address of an object

print(help(id))

=======================================

Add 2 number and return the result

def add():

return 100 + 500

# addition performed inside

print(add())

# Using return result we r getting outside a function

==============================

How many values u can return

Ans: n values

def add():

return 100 + 500

# addition performed inside

print(add())

#2

result = add()

print(result)

================================

def report():

check = '+ve'

if check == '+ve':

return 'admit'

else:

return 'HomeQ'

print(report())

step = report()

if step == 'admit':

print('check vacant beds')

else:

print('HomeQ')

==================================

How to return multiple values

a = 20

b = 50

def sample():

return a+b,a-b,a\*b,a/b

print(sample())

# unpack the results

add,sub,mul,div = sample()

print(add, sub,mul,div)

------------------------------

a = 20

b = 50

def sample():

return a+b,a-b,a\*b,a/b

print(sample())

# unpack the results

add,sub,mul,div = sample()

print('addition:',add,'subtraction:',sub,mul,div)

Function with parameters

def sample(a):

# a is a parameter/ an argument

print(a)

sample(10)

sample('python')

---------

How many parameters we can supply?

Ans: n

======================

def sample(a,b,c):

print(a)

print(b)

print(c)

sample(1,2,3)

# Rule: in declaration if we have 3 args then u must supply 3 values

# Rule: when we supply direct values these values are positional one

sample('A','B','C')

sample('depali','pune',411011)

# in values we can supply any type of data

=========================

def sample(a,b,c):

print(a,b,c)

sample('java','c','python')

===================

**Types of arguments**:

1. **Positional arguments:** follows sequence order

Positional argument means that the argument must be provided in a correct position in a function call.

Example:

def info(name,age,place):

print('name:',name)

print('age:',age)

print('place:',place)

info('Saurabh',30,'Satara')

# but if we change sequence then???

info(21,'Seema','Latur')

# in above case positional arg create a problem

# as we are not following a sequence

==========================================

**2. Keyword argument**: here sequence doesnt matter bcz we give a reference with args.  keyword argument is a function argument with a name label. Passing arguments as keyword arguments means order does not matter.

Example:

def info(name,age,place):

print('name:',name)

print('age:',age)

print('place:',place)

info('Saurabh',30,'Satara')

# now we r taking help of arguments itself to allocate values properly

info(age=21,name='Seema',place='Latur')

=================================

**3. Default argument**: in declaration itself we can set a value as a default value

Rule: if we have not give value for default arg then it will take default value, otherwise it will take the value given by a user.

Default argument should follow positional arguments.

==================

def info(name,place='Maharashtra'):

print(name,place)

info('Ankush')

info('Ganesh','Kolhapur')

=====================

# When non-default argument follows default argument

# its not allowed as per suntax

def info(place='Maharashtra',name):

print(name,place)

info('Ankush')

info('Ganesh','Kolhapur')

-------------------------

def show(p,q,r=400):

print(p,q,r)

# in above case p,q are positional args

# r is default arg.

# so r should always follow p,q

show(10,20)

------------------

def show(r=400,p,q): #not allowed

print(p,q,r)

# in above case p,q are positional args

# r is default arg.

# so r should always follow p,q

show(10,20)

============================

**4. Variable length argument:** multiple values u can supply

at the time of calling or u may keep calling empty

it has 2 types:

1. **Variable length positional args**

in order make a normal variable as a variable length use \* infront of that variable

Example:

def func(\*var):

------------------

def show(\*n):

print(n)

show(1)

show('py','c','c#')

show(-1,-2,-3,-4,-5,-6)

show()

---------------------

It returns tuple as an output

-----------------------

2. **Variable length keyword args**

in order make a normal variable as a variable length

keyword arg

use \*\* Infront of that variable

second imp things,

supply the values as keyword args

Example:

def func(\*\*var):

========================

def show(\*\*n):

print(n)

show(a=10,b=20)

show()

show(name='anil',age=45,place='pune')

It returns result in dict format

===========================

var len positional: \*args

def info(\*args):

print(args)

info()

info(1,2,3)

info('ramesh',33,45000)

--------------------------------

var len keyword: \*\*kwargs

def info(\*\*kwargs):

print(kwargs)

info()

info(p=1,q=2,r=3)

info(nm='ramesh',ag=33,s=45000)

============================

ASSIGNMENT:

What is difference between \*args and \*\*kwarg

\*args : 1

1. It returns tuple as an output

2. Its output is immutable in nature

3. positional argument maintain sequence

4. \*args specifies the number of non-keyworded arguments that can be passed and the operations that can be performed on the function in Python

\*\*kwargs:

1. It returns Dictionary as an output

2. Its output is mutable in nature

3. kwargs follows positional args

4. \*\*kwargs is a variable number of keyworded arguments that can be passed to a function that can perform dictionary operations

**Examples:**

Take inputs from use and supply to a function

---------------------------------

def sample(nm,age,sal):

print('Name:',nm,'Age:',age,'salary:',sal)

for i in range(3):

name = input('Enter the name:')

a = int(input('Enter the age:'))

salary = float(input('Enter the salary:'))

sample(nm=name, age=a, sal=salary)

print('--------------------------')

=============================

BANK APPLICATION

========================

balance = 0.0

name = input('Enter your name:')

print('Welcome', name, 'to SBI PUNE KATRAJ BRANCH')

print('You are fresh account holder')

print('Your current balance is Rs:', balance)

print('------------------')

print('We have following options')

print('1.Credit\n2.Debit\n3.Check Balance\n4.Exit')

while True:

def credit(amt):

global balance

balance+=amt

return balance

def debit(amt):

global balance

balance-=amt

return balance

ch = int(input('Enter your choice:'))

if ch == 1:

print(name,'You selected Credit option')

amt = float(input('Enter amount:Rs.'))

print('Account balance after Credit:Rs.',credit(amt))

elif ch == 2:

print(name,'You selected Debit option')

amt = float(input('Enter amount:Rs.'))

print('Account balance after Debit:Rs.',debit(amt))

=================================================

VVVIMP

Lambda function:

known as Anonymous function or Nameless function

It has same structure and property of a normal function but the purpose to introduce this is, whenever we want to write a function for temp. purpose and second imp thing is it reduces the code.

Syntax:

lambda parameter/s: expression

lambda: keyword

parameter: arguments

expression: operation [only ONE expression is allowed at a time]

=====================================

Normal func: for squaring the number

def square(num):

return num\*num,num+100

print(square(10))

-------------------------------------

# Using lambda function

sq = lambda num:num\*num

print(sq(25))

# lambda function has default return statement

print(square)

print(sq)

------------------------

Example:

# positional args

add = lambda x,y:x+y

print(add(1,2))

# keyword args

sub = lambda x,y:x-y

print(sub(y=100,x=200))

# default arg

conv = lambda nm='Guest':nm.upper()

print(conv())

print(conv('ramesh'))

#var len

var = lambda \*args:sum(args)

print(var(10,20))

print(var(1,2,3,4,5))

=================================

test = lambda x,y,z:x\*100+y\*10

print(test(1,2,3))

===============================

test = lambda x,y,z:(x\*100,y\*10,z+90)

print(test(1,2,3))

===================================

don = lambda nm='Shani':len(nm)

print(don())

print(don('Python'))