**Functions**

**Day-01**

**24-03-2025**

**=================**

Part-01

--------

What is Function?

Why Function?

How to define the function?

Various approaches to define the function

Types of function parameters

Recursion

Types of Variables/Scope

Part-02

-------

Nested Functions

Function Aliasing

Name less functions/Anonymous functions

Decorators

Generators

**Functions Part-01**

**============**

**1) What is function?**

**------------------------**

Ex:

id() ==> pre-defined function

return/give theof the specified data

type() ==> pre-defined function

can give the type of the data

-> function can be used to define the functionality or task.

-> To define or implement the task we need to define the block of code (may contains one statement or more than one statement)

-> The function is simply defined as it is a named block

which can be used to perform the specified task/functionality.

-> Functions can be classified into two types:

 1) Built-in functions

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

 -> also called as "Pre-defined functions".

 Ex: id(), len(), type() etc.

 2) User defined functions

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

 -> The functions can be defined as per the business requirements by the programmer are called as "User-defined functions".

**2) Why Functions?**

**----------------------**

-> Modular programming

Ex:

IRCTC:

 Book Train ticket

 Hotels

 Book Bus Tickets

 Flight Tickets

 etc.

-> Easy understandability of the code

functions can increase the readability.

-> We can able to execute the specific part of the code/application when we need and in any number of times.

Ex: Phonepe

Chat

Money Transfer

Check balance

E bills etc.

**3) How to define the function?**

**-----------------------------------**

-> def keyword can be used to define the functions.

Syntax:

 def function-name():

 function-body

 with statements

-> The function definition contains:

 1) function header

 2) function body

-> function header describe:

 1) name of the function

 2) data of the function

-> function body can include group of statements

 which can use to define/perform the task.

Note:

-----

-> When we have defined the function, the function must be invoked/called.

**4) Function call/Function Invoking:**

**------------------------------------------**

Syntax:

 function-name()

-> We can define the function call/function invoke statement always after the function definition.

Example for user-defined function:

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

def greetings():

 print("Hi")

 print("Good Evening")

 print("Welcome to Python class at Ashok IT.")

greetings()

**Various ways to define the functions:**

**--------------------------------------------**

-> there are three ways to define functions:

 1) functions without parameters/arguments

 2) functions with parameters/arguments

 3) functions with return type

parameters/arguments ==> the data to the functions called as "parameters/arguments".

1) function without parameters:

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

syntax:

 def function-name():

 function-body

**"""**

**Write a python program to generate all armstrong numbers from the given range**

**using user-defined functions.**

**"""**

def ArmstrongNumberFromRamge():

 r1 = int(input("Enter the start value:"))

 r2 = int(input("Enter the stop value:"))

 for i in range(r1, r2+1):

 s = 0

 n = i

 l = len(str(n))

 while n != 0:

 d = n % 10

 p = d \*\* l

 s = s + p

 n = n // 10

 if s == i:

 print(i,end = "\t")

ArmstrongNumberFromRamge()

2) functions with parameters/arguments

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

Syntax:

 def function-name(data-name1,data-name2,...):

 function-body

**# Write a python program to accept the sides of the triangle as an input**

**# and check what type of the triangle it can form using user-defned function.**

def checkTypeOfTriangle(side1,side2,side3):

 if side1 == side2 and side2 == side3:

 print("It can form An Equilateral Triangle.")

 elif side1 == side2 or side2 == side3 or side3 == side1:

 print("It can form An Isosceles Triangle.")

 else:

 print("It can form Scalene Triangle.")

# to send the data to the function, we have two aproaches:

# direct sending

checkTypeOfTriangle(9,7,8)

# indirect sending

side1 = int(input("Enter first side value:"))

side2 = int(input("Enter second side value:"))

side3 = int(input("Enter third side value:"))

checkTypeOfTriangle(side1,side2, side3)

3) functions with return type

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

return keyword:

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

return is one of the transfer statement.

-> return can use to return the value from the function by stopping its execution.

Syntax:

 return value

Example for return statement:

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

def addition(a,b):

 c = a + b

 return c

# when the function has return statement then, the function call must be assign to variable and

# make use for further

result = addition(10,20)

print("The Sum of two numbers = ",result)

# instead of taking another variable, we can directly define the function call in the print()

print("The Sum of Two numbers = ",addition(234,127))

**Day-02**

**25-03-2025**

**=====================**

**Function Parameters:**

**==============**

What is parameter?

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

-> the data/value can be defined for the functions are called as "Parameters".

-> also called as "Arguments".

-> Two types of parameters:

 1) Actual Arguments

 2) Formal Arguments

-> When we can define the arguments/parameters within the function call those are called as "Actual Arguments/Actual Parameters".

-> When we can define the arguments/parameters within the function definition are called as "Formal Arguments/Formal Parameters".

Types of Parameters:

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

-> The function parameters can define in multiple ways:

 1) Positional Arguments

 2) Keyword Arguments

 3) Default Arguments

 4) Variable Length Arguments

1) Positional Arguments:

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

-> The order of parameters what we can follow in the function call, the same order can follow for the substitution in the place of formal arguments (in the function header). This representation is called as "Positional Argument Representation".

Ex:

def function(a, b, c):

 print(a)

 print(b)

 print(c)

function(100,200,300)

p = 1000

q = 2000

r = 3000

function(r, p, q)

Example for Positional Arguments:

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

def person(name,age):

 print("The Details of the person = ")

 print("Name of the user = ",name)

 print("Age of the user = ",age)

person("Ravi",31)

person(31,"Ravi")

2) Keyword Arguments

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

-> Some times, we can send the data/values/parameters to the function with some name this representation is called as "Keyword Argument Representation".

Syntax:

 def function(par1, par2):

 function body

 action of statements

 function(par1 = val1, par2 = val2, ...)

Example for Keyword Argument representation:

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

def person(name,age):

 print("The Details of the person = ")

 print("Name of the user = ",name)

 print("Age of the user = ",age)

person(name = "Karthik",age = 32)

person(age = 32,name = "Kishore")

Q: Is it possible to define the function with positional arguments and keyword arguments?

Yes

def addition(n1,n2,n3,n4,n5):

 s = n1 + n2 + n3 + n4 + n5

 print("The sum = ",s)

a = int(input("Enter the first value:"))

b = int(input("Enter the second value:"))

c = int(input("Enter the third value:"))

d = int(input("Enter the forth value:"))

e = int(input("Enter the fifth value:"))

# addition(n5 = c,n2 = b,d,a,e) # positional args followed keyword args

addition(a,d,e,n5 = b,n4 = c) # keyword args followed positional

Here:

The keyword arguments must be followed the positional arguments

3) Default Arguments

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

Sometimes, the functions can execute without accepting the data from the function call even the function has defined with arguments. In this case, we can use "default arguments".

Syntax:

 def function-name(par1 = default-value, par2 = default-value,...):

 action of statements

 function-name()

Example for the Default Arguments:

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

def multiplication(a = 1,b = 2,c = 3,d = 2,e = 1):

 product = a \* b \* c \* d \* e

 return product

res1 = multiplication() # no values are sending

print("The Result after the multiplication = ",res1)

res2 = multiplication(121) # 121 can place in 'a'

print(res2)

res3 = multiplication(d = 132)

print(res3)

res4 = multiplication(27,19,d = 32,e = 26,c = 27)

print(res4)

4) Variable Length Arguments

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

When we want to execute the function with any number of actual parameters then, we can use "Variable Length Arguments".

Syntax:

 def function-name(\*args):

 function-body

 function-name()

 function-name(val1, val2,..)

-> In the function definition, \*args is used for variable number of parameters

\*args ==> with in the function

 \* args can behave as tuple.

tuple --> (v1, v2, v3,...)

Example for Variable Length Arguments:

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

def addition(\*parameters):

 s = 0

 for i in parameters:

 s = s + i

 print("The Sum of group of elements = ",s)

addition()

addition(10)

addition(121,132)

addition(11,22,33,44,55,66,77,88,99,110)

**Day-03**

**26-03-2025**

**====================**

**Variables and its Scope:**

**================**

-> there are two types of variables:

 1) Local variables

 2) Global variables

1) Local variables:

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

-> also called as "function variables"

the variables which can be defined within the function are called as "Local variables".

-> The local variables are available within the same function. The local variables have the function scope.

When we can try to access in outside of the function scope, then we can get a run-time error is called as "name error".

def localVariables(p,q):

 a = 123

 b = 321

 print("A = ",a)

 print("B = ",b)

 print("P = ",p)

 print("Q = ",q)

localVariables(100,200)

# print(a)

# print(b)

# print(p)

# print(q)

Note:

-----

The formal arguments are also have function scope only. That means we can access within the function but not in outside the function.

Q: When the memory can be created for the local variables?

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

-> Local variables can reserve in the heap memory.

The memory space can be created for the local variables after the function call execution.

Q: When the memory can be released for the local variables?

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

the memory for the local variables can be released/deleted once the execution of the function is completed.

2) Global variables:

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

we can define the variables at once

and can able to access in anywhere of the program.

PI = 3.14

radius = 7

def areaOfCircle():

 area = PI \* radius \* radius

 print("The area = ",area)

def circumference():

 circum = 2 \* PI \* radius

 print("The circumference = ",circum)

areaOfCircle()

circumference()

Here, in the above case the global variables have defined above to all the function definitions. Where if you want to define the global variables after the functions, yes it is possible but must be above to function call.

def areaOfCircle():

 area = PI \* radius \* radius

 print("The area = ",area)

def circumference():

 circum = 2 \* PI \* radius

 print("The circumference = ",circum)

PI = 3.14

radius = 7

areaOfCircle()

circumference()

-> Global variables available within the entire program. That means when the program can start the global variable can be get start and when the program can be closed then the memory of the global variable can be destroyed.

Q: Do we able to create global variables within the functions?

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

Yes.

we can create global variables within the function using "global" keyword.

i) within the function, we should declare the variable as global

Syntax:

 def function():

 global variable

 variable = value

# create the global variable within the function

def areaOfRectangle():

 global length

 length = 10

 global breadth

 breadth = 8

 area = length \* breadth

 print("The area of the rectangle = ",area)

def perimeter():

 perimeter = 2\*(length + breadth)

 print("The perimeter = ",perimeter)

areaOfRectangle()

perimeter()

Q: Do we able to modify global variables after the definition?

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

a = 100

b = 200

def f3():

 print("a = ", a)

 print("b = ", b)

def f1():

 global a

 a = 1000

 global b

 b = 2000

 print("a = ",a)

 print("b = ",b)

def f2():

 print("a = ",a)

 print("b = ",b)

f3()

f1()

f2()

# global variables and local variables are with the same name

# local variables are highly prioritised than global variables within the functions.

a = 100 # global variables

b = 200

def f1():

 print(a)

 print(b)

def f2():

 a = 300

 b = 400 # local variables

 print(a)

 print(b)

def f4():

 global a

 a = 1234

 global b

 b = 4321

def f3():

 print(a)

 print(b)

f1()

f2()

f4()

f3()