Python Functions is a block of statements that return the specific task.

##Syntax: Python Functions

def function\_name(parameters)

#statement

return expression

##Creating a python function

We can create a python function using def keyword

# A simple Python function

def fun():

print("Welcome to Hacktoberfest")

##Calling a Python Function

After creating a function we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

# A simple Python function

def fun():

print("Welcome to Hacktoberfest")

# Driver code to call a function

fun()

Output

Welcome to Hacktoberfest

##Defining and calling a function with parameters

#Syntax: Python Function with parameters

def function\_name(parameter: data\_type) -> return\_type:

"""Docstring"""

# body of the function

return expression

##Example

def add(num1: int, num2: int) -> int:

"""Add two numbers"""

num3 = num1 + num2

return num3

# Driver code

num1, num2 = 5, 15

ans = add(num1, num2)

print(f"The addition of {num1} and {num2} results {ans}.")

Output

The addition of 5 and 15 results 20

##More Examples

#some more functions

def is\_prime(n):

if n in [2, 3]:

return True

if (n == 1) or (n % 2 == 0):

return False

r = 3

while r \* r <= n

if n % r == 0:

return False

r += 2

return True

print(is\_prime(78), is\_prime(79))

output

False True

##Arguments of a Python Function

In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.

# A simple Python function to check

# whether x is even or odd

def evenOdd(x):

if (x % 2 == 0):

print("even")

else:

print("odd")

# Driver code to call the function

evenOdd(2)

evenOdd(3)

Output

even

odd

##Types of Arguments

Python supports various types of arguments that can be passed at the time of the function call.

Default arguments

# Python program to demonstrate

# default arguments

def myFun(x, y=50):

print("x: ", x)

print("y: ", y)

# Driver code (We call myFun() with only

# argument)

myFun(10)

Output

x: 10

y: 50

Keyword arguments

The idea is to allow the caller to specify the argument name with values so that caller does not need to remember the order of parameters.

# Python program to demonstrate Keyword Arguments

def student(firstname, lastname):

print(firstname, lastname)

# Keyword arguments

student(firstname='Kamalini', lastname='Das')

student(lastname='Das', firstname='Kamalini')

Output

Kamalini Das

Kamalini Das

##Example 1: Variable length non-keywords argument

# Python program to illustrate

# \*args for variable number of arguments

def myFun(\*argv):

for arg in argv:

print(arg)

myFun('Hello', 'Welcome', 'to', 'Hacktoberfest')

output

Hello

Welcome

to

Hacktoberfest

##Example 2: Variable length keyword arguments

# Python program to illustrate

# \*kwargs for variable number of keyword arguments

def myFun(\*\*kwargs):

for key, value in kwargs.items():

print("%s == %s" % (key, value))

# Driver code

myFun(first='Welcome', mid='to', last='Hacktoberfest')

Output

first == Welcome

mid == to

last == Hacktoberfest

##Docstring

The below syntax can be used to print out the docstring of a function

Syntax: print(function\_name.\_\_doc\_\_)

Example: Adding Docstring to the function

# A simple Python function to check

# whether x is even or odd

def evenOdd(x):

"""Function to check if the number is even or odd"""

if (x % 2 == 0):

print("even")

else:

print("odd")

# Driver code to call the function

print(evenOdd.\_\_doc\_\_)

Output

Function to check if the number is even or odd

##Return statement in python function

Syntax:

return [expression\_list]

Example: Python Function Return Statement

def square\_value(num):

"""This function returns the square

value of the entered number"""

return num\*\*2

print(square\_value(2))

print(square\_value(-4))

Output

4

16

##Pass by Reference or pass by value

# Here x is a new reference to same list lst

def myFun(x):

x[0] = 20

# Driver Code (Note that lst is modified

# after the function call.)

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

Output

[20, 11, 12, 13, 14, 15]

##When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken.

For example, consider the below program as follows:

def myFun(x):

# After below line link of x with previous

# object gets broken. A new object is assigned

# to x.

x = [20, 30, 4]

# Driver Code (Note that lst is not modified

# after the function call.)

1st = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

Output

[10, 11, 12, 13, 14, 15]

##Another example to demonstrate that the reference link is broken if we assign a new value (inside the function).

def myFun(x):

# After below line link of x with previous

# object gets broken. A new object is assigned

# to x.

x = 20

# Driver Code (Note that lst is not modified

# after the function call.

x = 10

myFun(x)

print(x)

Output

10

##Anonymous functions in Python Function

# Python code to illustrate the cube of a number

# using lambda function

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

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

print(cube(7))

print(cube\_v2(7))

Output

343

343

##Python Function within Functions

# Python program to

# demonstrate accessing of

# variables of nested functions

def f1():

s = 'I love Open Source'

def f2():

print(s)

f2()

# Driver's code

f1()

Output

I love Open Source

Example

#Program to illustrate

# the use of user-defined functions

def add\_numbers(x,y):

sum = x + y

return sum

num1 = 5

num2 = 6

print("The sum is", add\_numbers(num1, num2))

output

The sum is 11