

Module-3

Functions in Python

1. **Function:**

A function is a block of code that performs a specific task and can be called by other parts of the program. Functions are defined using the def keyword, followed by the function name and a set of parentheses that may contain parameters. The code block within the function is indented and is executed when the function is called.

* Functions help break our program into smaller and modular chunks for better readability.
* Information can be passed into a function as arguments.
* Parameters are specified after the function name inside the parentheses.
* We can add as many parameters as we want. Parameters must be separated with a comma.
* A function may or may not return data.
* In Python, a function is defined using the def keyword.

1. **Arguments Versus Parameters:**

* **Arguments:** An argument is a value that is sent to the function when it is called.

**Example:**

def multiply(a, b):

return a \* b

In this function, a and b are the parameters.

* **Parameters:** A parameter is the variable listed inside the parentheses in the function definition.

**Example:**

result = multiply(5, 10)

5 and 10 are the arguments passed to the function.

1. **Types of Functions in Python:**

* **Built-in Function:-** Python predefined functions that are readily available for use like min(), max(), sum(), print() etc.
* **User-Defined Functions:-** Function that we define ourselves to perform a specific task.
* **Anonymous Functions:** Function that is defined without a name. Anonymous functions are also called lambda functions. They are not declared with the def keyword.
* **Some Examples:**
* **Function to print - Welcome to Fingertips!**

def welcome():

print('Welcome to Fingertips!')

welcome()

**Output:**

Welcome to Fingertips!

* **Function to print - Square of a number(default parameter)**

def square(n=9):

n= n\*n

print(n)

square()

square(6)

**Output:**  
81

36

* **Function to print - Square of a number(pass an argument)**

def square(n):

n= n\*n

return n

square(8)

**Output:**

64

* **Function to print - Details of Student(pass 2 arguments)**

def stu\_details(name, course):

print('Name - ', name)

print('Course - ',course)

stu\_details('Ishan', 'Data Science')

**Output:**

Name - Ishan

Course - Data Science

* **Function to print - Details of Student(1 default argument, pass 2 argument)**

def stu\_details(organization='Fingertips', name, course):

print('Name - ', name)

print('Course - ',course)

print('Organization - ',organization)

stu\_details('Ishan', 'Data Science')

**Output:**

SyntaxError: non-default argument follows default argument

**Non-default arguments follow default argument:**

def stu\_details(name, course, organization='Fingertips'):

print('Name - ', name)

print('Course - ',course)

print('Organization - ',organization)

stu\_details('Ishan', 'Data Science')

**Output:**

Name - Ishan

Course - Data Science

Organization - Fingertips

* **Function to check whether a given number is even or odd.**

def even\_odd(num):

if num % 2 ==0:

print(num, ' is an even number')

else:

print(num, ' is an odd number')

even\_odd(3)

even\_odd(4)

**Output:**

3 is an odd number

4 is an even number

* **Function to concatenate strings(pass 3 arguments).**

def concat(name, age, course):

fullname = print(f'{name} is {age} years old and pursing {course} course at Fingertips.')

concat('Ishan', 22, 'Data Sciecne')

**Output:**

Ishan is 22 years old and pursuing Data Science course at Fingertips.

1. **Global and local variables**

**Global Variable:**

A global variable, on the other hand, is a variable that is declared outside of any function or block of code and can be accessed by any function or block in the program. Global variables are created when the program begins and are destroyed when the program exits. They can be accessed by any function or block in the program, regardless of where they were declared.

**Local Variable:**

A local variable is a variable that is declared within a function or block of code and can only be accessed within that function or block of code. It is created when the function or block is entered and is destroyed when the function or block is exited. Local variables are only accessible within the function or block they were declared in.

* **Variable with Global Scope can be accessed by all functions.**

var1 = 100 # Variable with Global scope.

def func1():

print(var1) # Value 100 will be displayed due to global scope of var1

func1()

print(var1)

**Output:**

100

100

* **Variable with Local Scope can't be accessed by other functions.**

def func1():

var2 = 10 # Variable with Local scope

print(var2)

def func2():

print(var2) # This will throw error because var2 has a local scope.

func1()

func2() #NameError: name 'var2' is not defined

**Output:**

10

20

* **Local variables are given preference over global variables.**

var1 = 100 # Variable with Global scope.

def func1():

var1 = 99 # Local scope(overwriting local variable)

print('Local variable -',var1)

func1()

print('Global variable -',var1)

**Output:**

Local variable - 99

Global variable - 100

* **Function to append items in a list.**

list1 = [10, 20, 30, 40]

def list\_app(list1):

list1.append(50)

print('List before calling function :',list1)

list\_app(list1)

print('List after calling funciton :',list1)

**Output:**

List before calling function : [10, 20, 30, 40]

List after calling funciton : [10, 20, 30, 40, 50]

* **Function to swap to numbers.**

def swap(num1, num2):

temp = num1

num1 = num2

num2 = temp

var1, var2 = num1, num2

print(f'Numbers after swapping are : {var1} and {var2}')

var1, var2 = 10, 20

print(f'Numbers before swapping are : {var1} and {var2}')

swap(var1, var2)

**Output:**

Numbers before swaping are : 10 and 20

Numbers after swaping are : 20 and 10

1. **args and kwargs**

In Python, args and \*kwargs are used as a way to pass a variable number of arguments to a function. These are used when we are not sure about the number of arguments at run-time or when we want to make customized argument function.

* \*args is used to pass a non-keyworded, variable-length argument list to a function.
* \*\*kwargs is used to pass a keyworded, variable-length argument list to a function.
* **args:**
* When we are not sure about the number of arguments being passed to a function then we can use \*args as function parameter.
* \*args allow us to pass the variable number of Non Keyword Arguments to function.
* We can simply use an asterisk \* before the parameter name to pass variable length arguments.
* The arguments are always passed as a tuple.
* We can rename it to anything as long as it is preceded by a single asterisk (\*).
* It's best practice to keep naming it args to make it immediately recognizable.

**Examples:**

1. **Function to print all the arguments.**

def my\_function(\*args):

for arg in args:

print(arg)

my\_function(1, 2, 3) #we can give any number of arguments.

**Output:**

1

2

3

1. **Function to sum any number of given arguments.**

def add(\*any\_num):

return sum(any\_num)

print('Sum is =',add(1,2,3))

print('Sum is =',add(1,2,3,4))

print('Sum is =',add(1,2,3,4,5))

print('Sum is =',add(1,2,3,4,5,6))

print('Sum is =',add(1,2,3,4,5,6,7))

**Output:**

Sum is = 6

Sum is = 10

Sum is = 15

Sum is = 21

Sum is = 28

* **kwargs:**
* \*\*kwargs allows us to pass the variable number of Keyword Arguments to the function.
* We can simply use an double asterisk \*\* before the parameter name to pass variable length arguments.
* The arguments are passed as a dictionary.
* We can rename it to anything as long as it is preceded by a double asterisk (\*\*).
* It's best practice to keep naming it kwargs to make it immediately recognizable.

**Examples:**

1. **Function to print all the details of a student.**

def Details(\*\*details):

print(details)

Details(Name='Ishan', Course='Data Science', Organization='Fingertips', State='Gujrat')

**Output:**

{'Name': 'Ishan', 'Course': 'Data Science', 'Organization': 'Fingertips', 'State': 'Gujrat'}

1. **Function to find sum of lists.**

def my\_function(\*\*kwargs):

for key, value in kwargs.items():

print(key, sum(value))

my\_function(a=[2,3,4], b=[5,6,7,8])

**Output:**

a 9

b 26

1. **Some Inbuilt Functions in Python**

There are many inbuilt functions in python:

* **Lambda Function:**
* A lambda function is a small anonymous function in Python.
* It can have any number of arguments but can only have one expression.
* The syntax for a lambda function is: lambda arguments: expression

**Examples:**

1. **Function to find square of a number using lambda function.**

square = lambda x: x\*x

print(square(5))

**Output:**

25

1. **Function to find even numbers from a list of given numbers.**

even\_numbers = lambda numbers: [x for x in numbers if x % 2 == 0]

print(even\_numbers([1, 2, 3, 4, 5])) # [2, 4]

**Output:**

[2, 4]

* **Filter Function**
* The filter() function in Python is used to filter a sequence of elements (e.g. a list, tuple, or string) based on a certain condition.
* It takes in two arguments: a function and an iterable.
* The function is applied to each element of the iterable, and if the function returns True for that element, it is included in the output.

**Examples:**

1. **Function to filter odd numbers from a list of numbers.**

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

odd\_numbers = filter(lambda x: x % 2 != 0, numbers)

print(list(odd\_numbers))

**Output:**

[1, 3, 5, 7, 9]

1. **Function to filter words having length>5.**

words = ["cat", "dog", "rabbit", "elephant", "giraffe"]

large\_words = filter(lambda x: len(x) >= 6, words)

print(list(large\_words))

**Output:**

['rabbit', 'elephant', 'giraffe']

* **Map Function:**
* The map() function in Python is used to apply a function to each element of an iterable (e.g. a list, tuple, or string) and return an iterator containing the results.
* It takes in two arguments: a function and an iterable.
* The function is applied to each element of the iterable, and the result of each function call is included in the output iterator.

**Examples:**

1. **Function to find squares of all numbers of a list**

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

sq\_numbers = map(lambda x: x\*x, numbers)

print(list(sq\_numbers))

**Output:**

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

1. **Function to convert all the words in upper-case.**

words = ["apple", "banana", "cherry", "date", "elderberry"]

uppercase\_words = map(lambda x: x.upper(), words)

print(list(uppercase\_words))

**Output:**

['APPLE', 'BANANA', 'CHERRY', 'DATE', 'ELDERBERRY']

* **Reduce Function:**
* The reduce() function in Python is a part of the functools module, it applies a function cumulatively to the elements of an iterable, from left to right, so as to reduce the iterable to a single value.
* It takes in two arguments: a function and an iterable.
* The function is applied to the first two elements of the iterable, then the result is passed as the first argument to the function along with the next element of the iterable and so on, until all the elements of the iterable have been processed.
* We have to import reduce from functools library.

**Examples:**

1. **Function to find product of all the numbers**

from functools import reduce

numbers = [1, 2, 3, 4, 5]

product = reduce(lambda x, y: x\*y, numbers)

print(product)

**Output:**

120

1. **Function to find largest number from a list of numbers.**

from functools import reduce

numbers = [1, 2, 3, 4, 5, 8, 9]

maximum = reduce(lambda x, y: x if x > y else y, numbers)

print(maximum)

**Output:**

9