Introduction to Python functions

Defining functions**:**

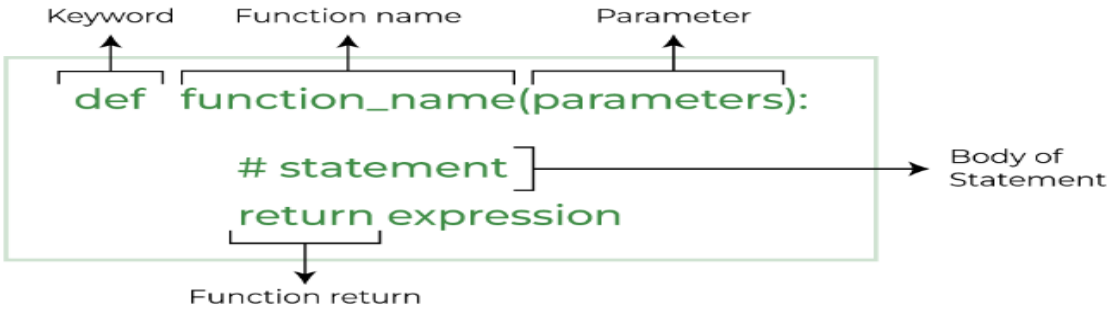
A function is a block of code that runs only when it is called. By using parameter/ arguments we can input data in to a function. A function can return result as a data.

There are two types of functions in python:

A. **Built – in functions:** These functions are the predefined functions that can be use as per the requirement.

B. **User defined functions:** These are the kind of function that's created by the user as per his real time requirement.

- **Creation and declaration of a user defined Function:** we can create a function in python by using the **def** keyword.




```
def    my_function():          # function name is my_function()
  print ("Hello i am here")
   my_function()                  # calling a function
```

- **Function arguments / parameters:**

  By using function argument, we can input the data to a function.

```
def my_function(coursename):
  print (coursename + " BTech")
my_function ("Civil")
my_function ("Electronics")
my_function ("Mechanical")
```

- **Arbitrary Arguments, *args :** This is the key feature in python, when we did not have confirmation about the parameter or how many parameter or arguments will be passes to a function then we can use this feature to declare a function.

```
def   my_function(*data):
                print ("The details is " + data[1])


          my_function ("Email", "Contact_no", "Address")
```

Function with Arguments / Parameters : By using function argument we can input the data to a functions.

```
def my_function(coursename):        # Function Arguments
 print (coursename + " BTech")
my_function ("Civil")
my_function ("Electronics")
my_function ("Mechanical")
```

- **Arbitrary Arguments, *args :** This is the key feature in python, when we did not have confirmation about the parameter or how many parameter or arguments will be passes to a function then we can use this feature to declare a function.

```
def   my_function(*data):
        print ("The details is " + data[1])
        my_function ("Email", "Contact_no", "Address")
```

- **Passing a List as an Argument:** We can send any data types of argument to a function like **string, number, list, dictionary etc.** It will be treated as the same data type inside the function.

```
def    my_function(course):
  for x in course:
   print(x)
   list_course = ["BTech", "MCA", "BCA"]
   my_function (list_course)
```

Output

```
BTech
MCA
BCA
```

- **<u>Return statement</u>**

  A return statement is written to exit the function and return the calculated value. A declared function will return an empty string if no return statement is written.

```
Def my_function(x):
  return 5 * x                  # Use of return statement
  print (my_function(3))
```

- **Pass Statement:** We cannot put function definition empty. But in exception case if we want to put function definition with no content then we can use **pass** statement. It will avoid the error.

```
def myfunction():
 pass
```

- **Anonymous Functions in Python**

  An anonymous function in Python is one that has no name when it is defined. We use the lambda keyword to define the anonymous functions

```
# using lambda function
cube_v2 = lambda x : x*x*x
print (cube_v2(7))
```

Functions vs Methods

Functions in Python	Methods in Python
Functions are created outside a class.	Methods are created inside a class.
Functions are not linked to anything.	Methods are linked with the classes they are created in.
Functions can be executed just by calling with its name.	To execute methods, we need to use either an object name or class name and a dot operator.
Functions can have zero parameters.	Methods should have a default parameter either self or cls to get the objects or class's address.
Functions cannot access or modify class attributes.	Methods can access and modify class attributes.
Functions are independent of classes.	Methods are dependent on classes.

For Example:

```
Class vegetables:
    def __init__(self, name):          # method1
        self.name = name
        self.veggies_eaten = 0
    def eat_one(self):                 # Method2
        self.veggies_eaten += 1
```

```
return "Eaten one {self.name}"
        v1 = vegetables("Radish")
        v2 = vegetables("Beetroot")
def eat_many(vegetable, count):   # Function
  for i in range(count):
    vegetable.eat_one()
        eat_many(v1, 20)
        eat_many(v2, 1)
        print(v1.veggies_eaten)
        print(v2.veggies_eaten)
```

Scope of variables

The scope of a variable is the location where we can declare a variable and access it if necessary. A variable can be found in the location where it was created, which is its scope.

Types of scope: In Python, there are two types of scope:

A.Local scope/local variables

A local scope or local variables are the variable which is created inside a function. This variable belongs to a local scope because it is only available inside the function it was created. The following example describe how to use local variable/ local scope:

```
# defining a function
# creating the variable x inside the function
def function1():
  x = 10
  print(x)

# calling the function
function1()
```

B.Global scope/global variable

A global scope/global variable is the variable which is created outside of a function. The function is created in the main body of the Python code, so it is called a global scope and the variable is referred to as a global variable.

```
# creation a global variabe
x = 10

# definig a function
def function1():
  print(x)

# calling the function
function1()

# printing x
print(x)
```

Python Lambda Functions

a lambda function is a special type of function without the function name. This function can have any number of arguments but only one expression, which is evaluated and returned.

Python Lambda Function Declaration:

Syntax: lambda argument(s): expression

Ex: 1

```
greet = lambda: print ('Hello World')          # declare a lambda function
greet()                                        # call lambda function
```

Ex2:

```
str1 = 'Cdacdelhi'
upper = lambda string: string.upper()          # use the upper function
print(upper(str1))
```

Ex3:

```
test = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list (filter (lambda x: (x % 2! = 0), test))
print(final_list)
```

Ex4:

```
x = lambda a: a + 10
print (x (5))                          # here X is the function
```

Ex5:

```
x = lambda a, b: a * b
print (x (5, 6))                       # here x is the function taking two arguments
```

Ex6:

```
x = lambda a, b, c: a + b * c
print (x(5, 6, 2))                     # here x is the function taking two arguments
```

Ex7:

```
item = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final = list(map(lambda x: x*2, item))
print(final)
Max = lambda a, b : a if(a > b) else b          # use of lambda function with if -else
print("The Result is:", Max(5, 34))
```

Lambda Function with filter ()

- The filter( ) method accepts two arguments in Python: a function and an iterable such as a list.
- The **filter** function can be applied to an iterable such as a list or a dictionary and create a new iterator.

Example:

```
age = [35, 12, 69, 55, 75, 14, 73]      # list is created for age
detail = list(filter( lambda num: (num % 2 != 0) , age ))
print('The list of age  is:',detail)
```

Lambda Function with map ( )

A method and a list are passed to Python's map() function.

Example: calculate the square of each number of a list using the map() function

```
numbers = [22, 24, 25, 21, 23, 37, 28, 29, 40]
new_list = list(map( lambda num: num ** 2 , numbers))
print( 'Square of number in the given list:' ,new_list )
```

```
# a list contains both even and odd numbers.
seq = [0, 1, 2, 3, 5, 8, 13,17,20,36,46,49,53]
# result contains odd numbers of the list
result = filter(lambda x: x % 2 != 0, seq)
print(“The odd numbers are”, list(result))
```

 # result contains even numbers of the list

result = filter(lambda x: x % 2 == 0, seq)

print(“ The Even Number are :” ,list(result))

Advantages of Lambda functions:

Conciseness: Lambda functions are one-liners, which can make code more concise and easier to read.

Readability: Lambda functions can encapsulate simple logic directly within the code, eliminating the need for separate function definitions. This can make your code more readable and easier to understand.

Flexibility: Lambda functions can be used in a variety of contexts, including as arguments to other functions, as part of list comprehensions, and as generators. This flexibility makes them a powerful tool for data manipulation and processing.

Speed: Lambda functions are generally faster than regular functions because they are implemented as anonymous functions and do not require a name lookup.

Disadvantage of Lambda Function:

- It can't perform multiple expressions.
- It can't contain any variable assignments (e.g., lambda x: x=0 will throw a SyntaxError).

Difference between Regular and Lambda function

Sr.No	Regular functions	lambda functions
1	Easy to interpret	Interpretation might be tricky
2	Can consists of any number of execution statements inside the function definition	The limited operation can be performed using lambda functions
3	To return an object from the function, return should be explicitly defined	No need of using the return statement
4	Execution time is relatively slower for the same operation performed using lambda functions	Execution time of the program is fast for the same operation
5	Defined using the keyword **def**	Defined using the keyword **lambda**

Built-In Functions: Python has a small set of extremely useful built-in functions.

1. **Type:** The type function returns the datatype of any arbitrary object. The possible types are listed in the types module.

   age = 30.5

   print(type(age))

2. **Str**

   horsemen = ['war', 'pestilence', 'famine']

   str(horsemen)

3. **dir**

   Python dir() function returns the list of names in the current local scope. If the object on which method is called has a method named __dir__(), this method will be called and must return the list of attributes.

   lang = ("C","C++","Java","Python")

   # Calling function

   att = dir(lang)

   # Displaying result

   print(att)

4. The python **len()** function is used to return the length (the number of items) of an object.

   Str1= 'Python'

5. **The max ()** function returns the item with the maximum value or the item with the maximum value in an iterable.

   print(len(str1))

   names_tuple = ('Chirag', 'Kshitiz', 'Dinesh', 'Kartik')

   print(max(names_tuple))

6. **id( ) function**

   The id() function returns a unique id for the specified object.

   names = ('Chirag', 'Kshitiz', 'Dinesh', 'Kartik')

7. **eva**l ( )

```
expression = 'x*(x+1)*(x+2)'
print(expression)
x = 3

result = eval(expression)
print(result)t(id(names))
```

8. **abs(): The abs() function returns the absolute value of a numeric argument.**

   the absolute values of two variables, positive and negative, are returned with the **abs()** function:

```
positive = 10
negative = -3.5
print(abs(positive))
print(abs(negative))
```

9. **.format()**

   The format() function returns a string from an input value, formatted to the provided specifications.

   Syntax: format(value, format_specification)

Specifier	Meaning
b	Binary format.
d	Decimal format.
e	Scientific format with lower case “e”.
E	Scientific format with upper case “e”.
f	Fixed-point format.
g	General format.
x	Hex format, lower case.
X	Hex format, upper case.

   Example:

```
pi = 3.14159
formatted = format (pi, '.3f')
print(formatted)
```

   Example:

```
a = 34
```

```
formatted = format (a, 'd')
print(formatted)
```

10. input()

the built-in input() function prompts the user for data that is converted to and returned as a string.

```
Syntax: input(prompt_string)
```

11. zip()

Takes multiple sequence data types as input and returns a single zip object made up of a list of tuples**. We can use** any list, tuple, set.

Note: dictionary is not possible to be uses in Zip function.

Syntax: zip(iterator1, iterator2,iterator3 ...)

```
animal = ['cat', 'dog', 'bird', 'great white shark']
weight = [9, 50, 0.33, 2000]
food =('Bread','Grass','Wheat','fish')
country={'India',"Srilanka",'Africa','USA'}
#disct={'leg':4, 'color': 'Black' , 'Nature': 'Runn'}
detail = zip(animal, weight,food,country)
print(list(detail))
```

12. Creating an user defined list using input function:

```
# creating an empty list
test = []
n = int (input("Enter number of elements : "))    # number of elements as input
for i in range(0, n):    // using for loop
        item = int(input())
        test.append(item)  # adding the element
print("The user input List is: ",test)

#list as input from user in Python Using map()
n = int(input("Enter number of elements : "))  # number of elements
# read inputs from user using map() function
```

```
a = list(map(int,input("\nEnter the numbers : ").strip().split()))[:n]
print("\nList is - ", a)
```

Concepts of Modules

Modules are simply files with the ". py" extension containing Python code that can be imported inside another Python Modules Operations Program. Python standard library contains well over 200 modules. Pandas,Scipy,seaborn, datetime, logging, math, numpy, os, pip, sys, and time etc.

There are two main types of Python modules: built-in modules and user-defined modules.

Advantages of modules –

Reusability: Working with modules makes the code reusability a reality.

Simplicity: Module focuses on a small proportion of the problem, rather than focusing on the entire problem.

Scoping: A separate namespace is defined by a module that helps to avoid collisions between identifiers.

Python Built-in modules : There are several built-in modules in Python, which you can import .

```
# importing built-in module math
import math
# using square root(sqrt) function contained
print(math.sqrt(25))
# using pi function contained in math module
print(math.pi)
# 2 radians = 114.59 degrees
print(math.degrees(2))
# 60 degrees = 1.04 radians
print(math.radians(60))
# Sine of 2 radians
print(math.sin(2))
# Cosine of 0.5 radians
```

```
print(math.cos(0.5))
# Tangent of 0.23 radians
print(math.tan(0.23))

# 1 * 2 * 3 * 4 = 24
print(math.factorial(4))
```

Creating user-defined module:

Installing Python Modules with Pip

Syntax: pip install <module name>

Ex: pip install numpy