The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

A python function consists of the following features:

1.Keyword def that marks the start of the function header.

2.A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.

3.Parameters (arguments) through which we pass values to a function. They are optional.

4.A colon (:) to mark the end of the function header.

5.Optional documentation string (docstring) to describe what the function does.

6.One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).

7.An optional return statement to return a value from the function.

##How to call a function in python?

Once we have defined a function, we can call it from another function, program, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Note: In python, the function definition should always be present before the function call. Otherwise, we will get an error.

We can use return type of the function and data type of the arguments in python.

##Arguments of a Python Function

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

#Types of Arguments

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

#Default arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

#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.

#Variable-length argument

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

\*args (Non-Keyword Arguments)

\*\*kwargs (Keyword Arguments)

##Docstring

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function.

The use of docstring in functions is optional but it is considered a good practice. Unless you can remember what you had for dinner last week,

always document your code. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as the

\_\_doc\_\_ attribute of the function.

##Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible

from outside the function. Hence, they have a local scope.The lifetime of a variable is the period throughout which the variable exists in the memory.

The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

On the other hand, variables outside of the function are visible from inside. They have a global scope.

We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.

##Return statement in Python function

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution.

If none of the above value is present with the return statement a None object is returned.

##Pass by Reference or pass by value.

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken.

##Anonymous functions in Python Function

In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

##Python Function within Functions

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of

the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

##Python Built-in Functions

Python has several functions that are readily available for use. These functions are called built-in functions.

Python abs()

returns absolute value of a number

Python all()

returns true when all elements in iterable is true

Python any()

Checks if any Element of an Iterable is True

Python ascii()

Returns String Containing Printable Representation

Python bin()

converts integer to binary string

Python bool()

Converts a Value to Boolean

Python bytearray()

returns array of given byte size

Python bytes()

returns immutable bytes object

Python callable()

Checks if the Object is Callable

Python chr()

Returns a Character (a string) from an Integer

Python classmethod()

returns class method for given function

Python compile()

Returns a Python code object

Python complex()

Creates a Complex Number

Python delattr()

Deletes Attribute From the Object

Python dict()

Creates a Dictionary

Python dir()

Tries to Return Attributes of Object

Python divmod()

Returns a Tuple of Quotient and Remainder

Python enumerate()

Returns an Enumerate Object

Python eval()

Runs Python Code Within Program

Python exec()

Executes Dynamically Created Program

Python filter()

constructs iterator from elements which are true

Python float()

returns floating point number from number, string

Python format()

returns formatted representation of a value

Python frozenset()

returns immutable frozenset object

Python getattr()

returns value of named attribute of an object

Python globals()

returns dictionary of current global symbol table

Python hasattr()

returns whether object has named attribute

Python hash()

returns hash value of an object

Python help()

Invokes the built-in Help System

Python hex()

Converts to Integer to Hexadecimal

Python id()

Returns Identify of an Object

Python input()

reads and returns a line of string

Python int()

returns integer from a number or string

Python isinstance()

Checks if a Object is an Instance of Class

Python issubclass()

Checks if a Class is Subclass of another Class

Python iter()

returns an iterator

Python len()

Returns Length of an Object

Python list()

creates a list in Python

Python locals()

Returns dictionary of a current local symbol table

Python map()

Applies Function and Returns a List

Python max()

returns the largest item

Python memoryview()

returns memory view of an argument

Python min()

returns the smallest value

Python next()

Retrieves next item from the iterator

Python object()

creates a featureless object

Python oct()

returns the octal representation of an integer

Python open()

Returns a file object

Python ord()

returns an integer of the Unicode character

Python pow()

returns the power of a number

Python print()

Prints the Given Object

Python property()

returns the property attribute

Python range()

returns a sequence of integers

Python repr()

returns a printable representation of the object

Python reversed()

returns the reversed iterator of a sequence

Python round()

rounds a number to specified decimals

Python set()

constructs and returns a set

Python setattr()

sets the value of an attribute of an object

Python slice()

returns a slice object

Python sorted()

returns a sorted list from the given iterable

Python staticmethod()

transforms a method into a static method

Python str()

returns the string version of the object

Python sum()

Adds items of an Iterable

Python super()

Returns a proxy object of the base class

Python tuple()

Returns a tuple

Python type()

Returns the type of the object

Python vars()

Returns the \_\_dict\_\_ attribute

Python zip()

Returns an iterator of tuples

Python \_\_import\_\_()

Function called by the import statement

##What are user-defined functions in Python?

Functions that we define ourselves to do certain specific task are referred as user-defined functions. The way in which we define and call functions

in Python are already discussed.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed

as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function

to someone else.

#Advantages of user-defined function

User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.

If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.

Programmers working on large project can divide the workload by making different functions.