What is a function in Python?

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Syntax of Function

```
def dunction_name(parameters)
    " " " doc string """
    statement(s)
```

Above shown is a function definition that consists of the following components.

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.

Example of a function

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

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.

```
greet('Paul')
```

Hello, Paul. Good morning!

Try running the above code in the Python program with the function definition to see the output.

```
def greet(name):
  """ This function greets to
```

```
    the person passed in as
    a parameter """“
    print("Hello, " + name + ". Good morning!")
greet('Paul')
```

Output

```
Hello, Paul. Good morning!
```

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

```
# function call
greet('Paul')
# function definition
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
# Error: name 'greet' is not defined
```

Docstrings

The first string after the function header is called the docstring and is short for documentation string. It is briefly used to explain what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. 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.

For example:

Try running the following into the Python shell to see the output.

print(greet.__doc__)

```
This function greets to

the person passed in as

a parameter
```

The return statement

The return statement is used to exit a function and go back to the place from where it was called.

Syntax of return

return [expression_list]

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

For example:

print(greet("May"))

Hello, May. Good morning!

None

Here, None is the returned value since greet( ) directly prints the name and no return statement is used.

Example of return

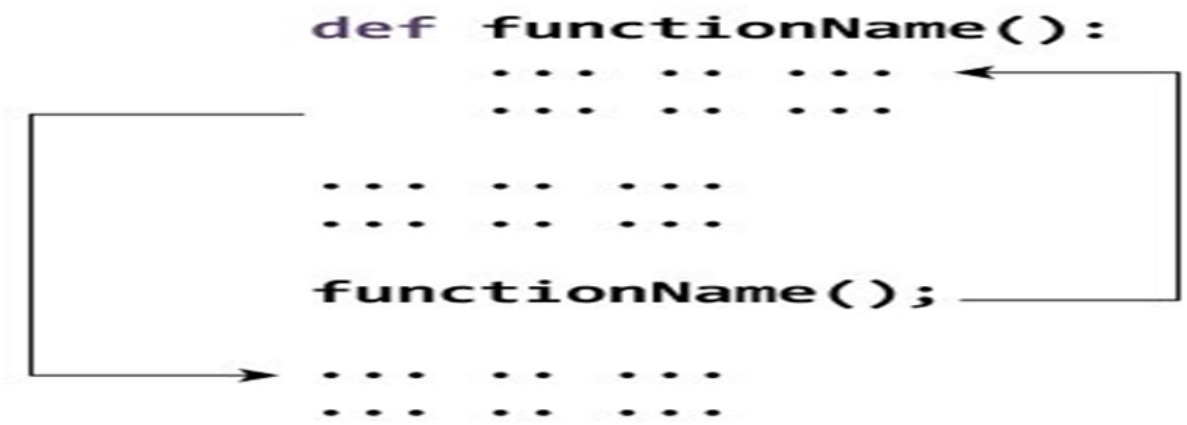
```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""
    if num >= 0:
    return num
    else:
    return -num
print(absolute_value(2))
print(absolute_value(-4))
```

Output

2

4

How Function works in Python?



Working of functions in Python

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.

Here is an example to illustrate the scope of a variable inside a function.

```
def my_func():
  x = 10
  print("Value inside function:",x)
x = 20
my_func()
print("Value outside function:",x)
```

Output

Value inside function: 10

Value outside function: 20

Here, we can see that the value of x is 20 initially. Even though the function my_func() changed the value of x to 10, it did not affect the value outside the function.

This is because the variable x inside the function is different (local to the function) from the one outside. Although they have the same names, they are two different variables with different scopes.

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.

Types of Functions

Basically, we can divide functions into the following two types:

1.Built-in functions - Functions that are built into Python.

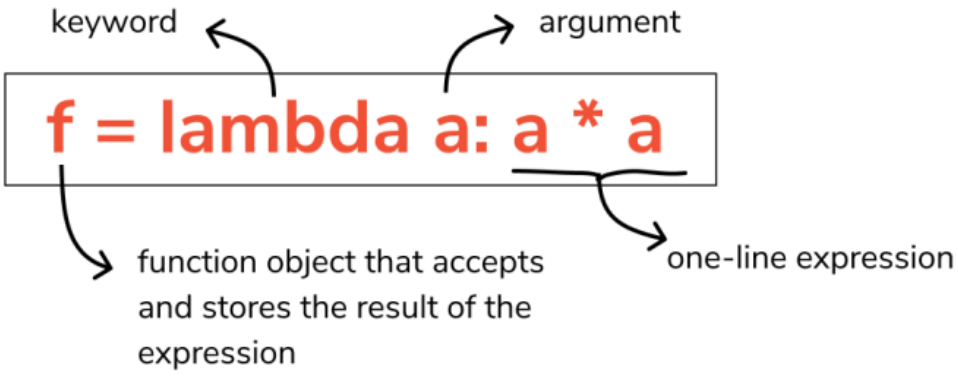
2.User-defined functions - Functions defined by the users themselves.

Advance Functions

Lambda Function

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.



Map Function

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

Syntax :

map(fun, iter)

fun : It is a function to which map passes each element of given iterable.

iter : It is a iterable which is to be mapped.

```
# Python program to demonstrate working
# of map.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
```

```
result = map(addition, numbers)

print(list(result))
```

Filter Function

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

syntax:

filter(function, sequence)

function: function that tests if each element of a

sequence true or not.

sequence: sequence which needs to be filtered, it can

be sets, lists, tuples, or containers of any iterators.

```
# function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False


# sequence
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

Generator Function

A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

The difference between the yield and the return statement in Python is:

Return. A function that returns a value is called once. The return statement returns a value and exits the function altogether.

Yield. A function that yields values, is called repeatedly. The yield statement pauses the execution of a function and returns a value. When called again, the function continues execution from the previous yield. A function that yields values is known as a generator.

```
# A generator function that yields 1 for first time,
# 2 second time and 3 third time
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3
```

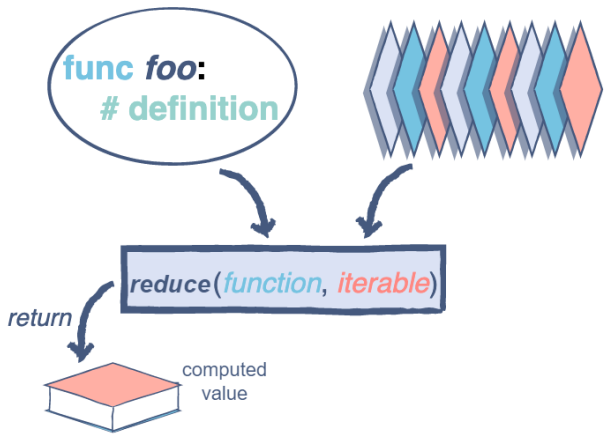
```
# Driver code to check above generator function

for value in simpleGeneratorFun():

        print(value)
```

Reduce Function

the reduce() function receives two arguments, a function and an iterable. However, it doesn't return another iterable, instead it returns a single value.



```
# python code to demonstrate working of reduce()

# importing functools for reduce()
import functools

# initializing list
lis = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a, b: a+b, lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lis))
```