Python Functions

- In Python, function is a group of related statements that perform a specific task.
- 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 code reusable.

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.

Syntax:

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

Above shown is a function definition which consists of following components.

1. Keyword def marks the start of function header.
2. Parameters (arguments) through which we pass values to a function. They are optional.
3. A colon (:) to mark the end of function header.
4. Optional documentation string (docstring) to describe what the function does.
5. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
6. An optional return statement to return a value from the function.

Example:

In [1]:
```
def print_name(name):
    """
    This function prints the name

    """
    print("Hello " + str(name))
```

Function Call

- 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.

```
In [2]: print_name("Students")
```

```
Hello Students
```

Docstring

- The first string after the function header is called the docstring and is short for documentation string.
- Although optional, documentation is a good programming practice, always document your code
- Doc string will be written in triple quotes so that docstring can extend up to multiple lines

```
In [3]: print(print_name.__doc__) # print doc string of the function
```

```
    This function prints the name
```

return Statement

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

Syntax:

```
return [expression_list]
```

-> return statement can contain an expression which 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 None Object.

In [4]:
```
def get_sum(lst):
    """
    This function returns the sum of all the elements in a list
    """
    #initialize sum
    _sum = 0

    #iterating over the list
    for num in lst:
        _sum += num
    return _sum
```

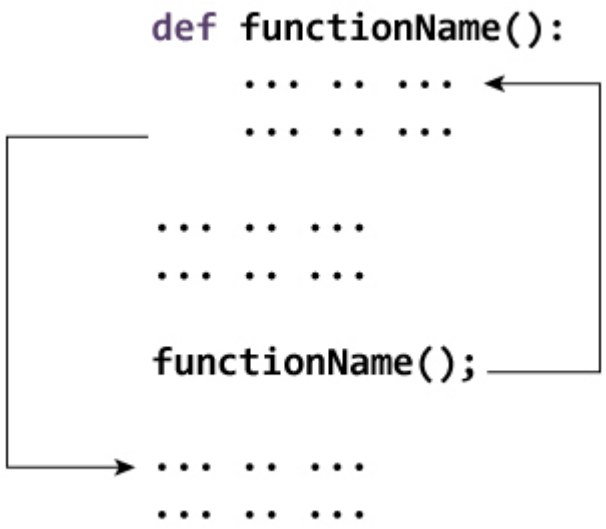
In [5]:
```
s = get_sum([1, 2, 3, 4])
print(s)
```

```
10
```

In [6]:
```
#print doc string
print(get_sum.__doc__)
```

```
    This function returns the sum of all the elements in a list
```

How Function works in Python?



Scope and Lifetime of variables

-> Scope of a variable is the portion of a program where the variable is recognized

-> variables defined inside a function is not visible from outside. Hence, they have a local scope.

-> Lifetime of a variable is the period throughout which the variable exits in the memory.

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

-> Variables are destroyed once we return from the function.

Example 1:

In [7]:
```
global_var = "This is global variable"

def test_life_time():
    """
    This function test the life time of a variables
    """
    local_var = "This is local variable"
    print(local_var)       #print local variable local_var

    print(global_var)      #print global variable global_var



#calling function
test_life_time()

#print global variable global_var
print(global_var)

#print local variable local_var
print(local_var)
```

```
This is local variable
This is global variable
This is global variable

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-d5226680661e> in <module>()
     19
     20 #print local variable local_var
---> 21 print(local_var)

NameError: name 'local_var' is not defined
```

Example 2:

In [ ]:
```
def my_func():
        x = 10
        print("Value inside function:",x)

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

Python program to print Highest Common Factor (HCF) of two numbers

In [ ]:
```
def computeHCF(a, b):
    """
    Computing HCF of two numbers
    """
    smaller = b if a > b else a  #consice way of writing if else statement

    hcf = 1
    for i in range(1, smaller+1):
        if (a % i == 0) and (b % i == 0):
            hcf = i
    return hcf

num1 = 98
num2 = 78

print("H.C.F of {0} and {1} is: {2}".format(num1, num2, computeHCF(num1, num2
)))
```

Python Program to Make a Simple Calculator

In [9]:
```
''' Program make a simple calculator that can add, subtract, multiply and di
vide using functions '''

# This function adds two numbers
def add(x, y):
   return x + y

# This function subtracts two numbers
def subtract(x, y):
   return x - y

# This function multiplies two numbers
def multiply(x, y):
   return x * y

# This function divides two numbers
def divide(x, y):
   return x / y

print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

# Take input from the user
choice = input("Enter choice(1/2/3/4):")

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

if choice == '1':
   print(num1,"+",num2,"=", add(num1,num2))

elif choice == '2':
   print(num1,"-",num2,"=", subtract(num1,num2))

elif choice == '3':
   print(num1,"*",num2,"=", multiply(num1,num2))

elif choice == '4':
   print(num1,"/",num2,"=", divide(num1,num2))
else:
   print("Invalid input")
```

```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4):4
Enter first number: 99
Enter second number: 43
99 / 43 = 2.302325581395349
```