Functions

- A function is **a collection of statements grouped together that performs a specific operation.**
- It is a block of code that is **used to achieve modularity and code reusability.**
- Once defined, **it can be called repeatedly from anywhere in the program** without re-writing the code again
- User can pass data, **known as parameters or arguments** into a function.
- A function can return data as a result.

Types of Functions

Basically, functions are of the following two types:

A) User-defined functions

B) Built-in functions

A) User-Defined Functions

- A function created by the user **while writing the program to achieve some tasks as per the programmer's requirement is called a user-defined function.**
- **Syntax:**

```
def function_name(parameters):
        """docstring"""
        Statement1
        Statement2
        .
        return [expression]
```

Rules to declare and define a user-defined function:

- Function blocks begin with the **def** keyword**.**
- The **def** keyword is followed by the **function name and parentheses ( ) and colon (:) sign.**
- Any input **parameters or arguments** s**hould be placed within these parentheses. Parameters are optional** i.e. a function may not have any parameters.
- The first statement of a function can be an **optional statement** – which is called as the **docstring** (which are enclosed in triple single quotes or triple double quotes).
- The **return statement exits a function optionally,** passing back an expression to the caller. A return statement with no arguments is the same as return None.

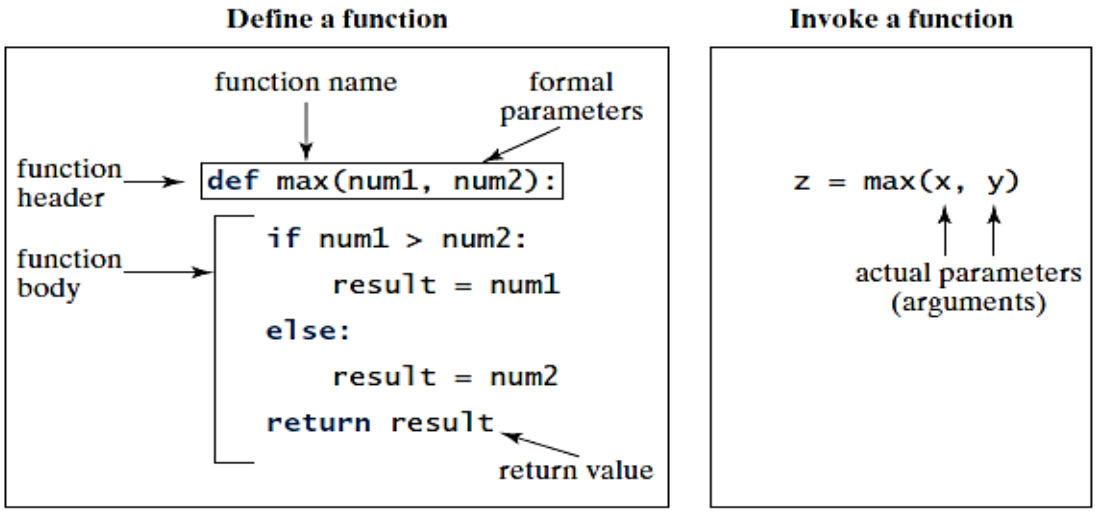
Note:

Docstring

- The **first statement in the function body can be a string 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.
- All functions should have a docstring.
- **Declaring Docstrings:** The docstrings are declared using "'triple single quotes'" or """triple double quotes""" just below the class, method or function declaration.

Creating a Function

- A function contains a header and body.
- The *header* begins with the **def** keyword, followed by the function's name and parameters, and ends with a colon.



Example:

```
def my_function():
  print("Hello from a function")
```

Defining a Function

- In a function's definition, the user defines what task to be performed.
- The function body contains a collection of statements that define what the function does.

Example:

```
# Function definition
def print_me( str ):
   print str
   return;
```

The pass Statement

- Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

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

Calling a Function

- **Calling a function executes the code in the function.**
- The program that calls the function is called a caller. When a program calls a function, program control is transferred to the called function.
- A called function returns control to the caller when its return statement is executed or the function is finished.
- To call or invoke a function, use the function name followed by parenthesis:

Example:

```
#To call printme function

print_me("I'm first call to user defined function!")
print_me("Again second call to the same function")
```

Output:

```
I'm first call to user defined function!
Again second call to the same function
```

Return Statement

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

Syntax:

```
return statement
```

Example:

```
def square_value(num):
  """This function returns the square value of the entered number"""
  return num**2

print(square_value(2))
print(square_value(-4))
```

Output

```
4

16
```

Example:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:
15
25
45

Example:

```
def sum( arg1, arg2 ):
   total = arg1 + arg2
   print "Inside the function : ", total
   return total;

# Call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

Output:
Inside the function: 30
Outside the function: 30

Example

```
def add_numbers(x,y):
   “sum”
   sum = x + y
   return sum
num1 = 5
num2 = 6
print("The sum is", add_numbers(num1, num2))
```

Output
The sum is 11

Example: Input From Console

```
# function definition

def addnum():
  fnum = int(input("Enter first number: "))
  snum = int(input("Enter second number: "))
  sum = fnum + snum
  print("The sum of ",fnum,"and ",snum,"is ",sum)
#function call
addnum()
```

Output:
Enter first number: 5
Enter second number: 6
The sum of 5 and 6 is 11

Note:

- Every function in Python returns a value whether you use **return** or not.
- If a function does not return a value, by default, it returns a special value **None**.
- For this reason, a function that does not return a value is also called a **None** *function*.
- The **None** value can be assigned to a variable to indicate that the variable does not reference any object.
- A **return** statement is not needed for a **None** function, but it can be used for terminating the function and returning control to the function's caller.
- Syntax

  return

  or

  return None

Function Arguments

- Arguments are the **values passed inside the parenthesis of the function call** which is received in corresponding parameter defined in function header.
- The names of the arguments used in the definition of the function are called formal arguments/parameters.
- Objects actually used while calling the function are called actual arguments/parameters.

Example:

```
# A simple Python function to check whether x is even or odd
 def evenOdd(x):
   if (x % 2 == 0):
      print("even")
   else:
      print("odd")

# Function call
evenOdd(2)
evenOdd(3)
```

Output

even
odd

Note:

- The terms *parameter* and *argument* can be used for the same thing
- From a function's perspective:
  - ✓ A parameter is the variable listed inside the parentheses in the function definition.
  - ✓ An argument is the value that is sent to the function when it is called.

Types of Arguments

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

1) **Default arguments/ Non-Arguments**
2) **Keyword arguments**
3) **Arbitrary Arguments /Variable-length arguments/ Non-Keyword Arguments**
4) **Arbitrary Keyword Arguments/Variable-length Keyword Arguments**

1. Default arguments

A function **assumes a default value**, **if a value is not provided in the function call** is called a function with default argument.

Example:

```
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)


myFun(10)
```

Output

```
x:  10
y:  50
```

Example:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

2. Keyword arguments

A function that is called by using keywords in the argument such that the **caller does not need to remember the order of parameters.**

Example:

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)
# Keyword arguments
student(firstname='Alice', lastname='Bond')
student(lastname='Bond', firstname='Alice')
```

Output

```
Alice Bond
Alice Bond
```

Example:

```
def multiply(a, b):
    return a*b
print(multiply(a = 10, b = 5))
print(multiply(b = 20, a = 9))
```

Output

```
50
80
```

3. Arbitrary Arguments / Variable-length arguments/ Non-Keyword Arguments (*args)

- If a caller does not know how many arguments that will be passed into a function, an asterisk sign * before the parameter name in the function definition is used.
- This way the function will receive multiple arguments.

Example:

```
def myFun(*parameters):
    for a in parameters:
    print(a)
myFun('Hello', 'Welcome', 'to', 'Data Science')
```

Output

```
Hello
Welcome
to
Data Science
```

Example:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Output:
The youngest child is Linus.

4. Arbitrary Keyword Arguments / Variable-length Keyword arguments (**args)

- If a caller does not know how many arguments that will be passed into a keyword argument function, two asterisk: ** before the parameter name in the function definition is used.
- This way the function will receive multiple keyword arguments.

Example:

```
def greet(**person):
        print('Hello ', person['firstname'],  person['lastname'])

greet(firstname='Steve', lastname='Jobs')
greet(lastname='Jobs', firstname='Steve')
greet(firstname='Bill', lastname='Gates', age=55)
```

Output:

```
Hello Steve Jobs
Hello Steve Jobs
Hello Bill Gates
```

Extra Points:

Example for Default argument and Keyword argument

Consider a student function:

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')
```

If the function is called without the argument, the argument gets its default value. The function *student* contains 3-arguments out of which 2 arguments are assigned with default values. So, the function *student* accepts one required argument (*firstname*), and rest two arguments are optional.

Example: Calling functions with default arguments

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')


# 1 positional argument
student('John')

# 3 positional arguments
student('John', 'Gates', 'Seventh')

# 2 positional arguments
student('John', 'Gates')
student('John', 'Seventh')
```

Output:

```
John Mark studies in Fifth Standard
John Gates studies in Seventh Standard
John Gates studies in Fifth Standard
John Seventh studies in Fifth Standard
```

In the first call, there is only one required argument and the rest arguments use the default values. In the second call, *lastname* and standard arguments value is replaced from default value to new passing value. We can see the order of arguments is important from the 2nd, 3rd, and 4th calls of the function.

Example: Calling functions with keyword arguments

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')


# 1 keyword argument
student(firstname ='John')


# 2 keyword arguments
student(firstname ='John', standard ='Seventh')


# 2 keyword arguments
student(lastname ='Gates', firstname ='John')
```

Output:

John Mark studies in Fifth Standard

John Mark studies in Seventh Standard

John Gates studies in Fifth Standard

In the first call, there is only one required keyword argument.
In the second call, one is a required argument and one is optional(standard), whose value gets replaced from default to a new passing value.
In the third call, we can see that **order in keyword argument is not important.**

Some Invalid function calls

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
     print(firstname, lastname, 'studies in', standard, 'Standard')

# required argument missing
student()

# non keyword argument after a keyword argument
student(firstname ='John', 'Seventh')

# unknown keyword argument
student(subject ='Maths')
```

The above code will throw an error because:

- In the first call, value is not passed for parameter *firstname* which is the required parameter.
- In the second call, there is a non-keyword argument after a keyword argument.
- In the third call, the passing keyword argument is not matched with the actual keyword name arguments.