Python – Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task. To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or can not return one or more values.

There are three types of functions in Python:

- Built-in functions, such as help() to ask for help, min() to get the minimum value, print() to print an object to the terminal,… You can find an overview with more of these functions here.
- User-Defined Functions (UDFs), which are functions that users create to help them out.
- Anonymous functions, which are also called lambda functions because they are not declared with the standard def keyword.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.Function blocks begin with the keyword def followed by the function name and parentheses (( )).Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

CONFIDENTIAL

The first statement of a function can be an optional statement - the documentation string of the function or docstring.The code block within every function starts with a colon (:) and is indented.The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax:

Def functionname(parameters):

"function_docstring"

function_suite

return [expression]

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function .

CONFIDENTIAL

For example :-

```
# Function definition is here
defprintme( str ):
    "This prints a passed string into this function"
    printstr
    return

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Pass by reference

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function**.**

CONFIDENTIAL

For example :-

```
# Function definition is here
defchangeme( mylist ):
   "This changes a passed list into this function"
mylist.append([1,2,3,4]);
print "Values inside the function: ", mylist
return

# Now you can call changeme function
mylist = [10,20,30];
changeme(mylist );
print "Values outside the function: ", mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

CONFIDENTIAL