Because learning changes everything.®

TITLE: Python Programming: Problem Solving, Packages and Libraries

Lecture PPT

Edition

Learning Objectives

- Understand the concept of function.
- Identify the three types of functions: (i) Built-in functions (built into the Python interpreter) (ii) Functions imported from modules and (iii) Functions created/ written by the programmer.
- Differentiate between “writing a function” and “calling a function”.
- Use the “built-in” functions from modules, that is, math and random.
- Describe the syntax of a function and the “flow of execution” in a function call.
- Explain how “parameters” are passed to a function and differentiate between “parameters” and “arguments”.
- Outline the concepts of “scope”, “namespace” and “lifetime” of a variable in a function call.
- “Combine” functions through “function composition”.

What are Python Functions?

A function can be defined as follows:

- **A function is a “named sequence of statement(s)”.**
- **It contains some line of code(s), which are executed sequentially (that is, one after another) from the top to the bottom.”**
- **It can perform computations.**

This definition can be explained as follows:

- **Suppose you have a block of code which you want to execute.**
- **You can name it by a function name and use or reuse it in a script.**
- **Hence, a function is a technique whereby you can form a group or block of statements and give them with a function name.**

Functions can also do the following two things:

- **They can take parameters and**
- **They can return a result or a value or even an object.**

Some Built-in functions

(Python has a number of built-in functions. Some are discussed in following slides)

abs(x)

This function:

- **Returns the absolute value of a number.**
- **The argument to this function may be an integer or a floating point number.**
- **If a complex number is given as argument, then the function returns its magnitude.**

Example:

```
>>> abs(-9)
9
>>> abs(3 + 4j) # abs is under root of 3 square plus 4 square
5.0
>>> abs(-3 -4j) # abs is under root of -3 square and -4 square
5.0
```

bool([x])

The use of square brackets indicates that the parameter is optional. If you don’t give a parameter to the bool() function, it will return a False.

The use of square brackets indicates that the parameter is optional. If you don’t give a parameter to the bool() function, it will return a False.

This function does the following:

- **It “converts” a value to a Boolean equivalent;**
- **In the “conversion”, it uses the standard truth testing procedure;**
- **If x is either False or is not given, then a value of False is returned; otherwise a value of True is returned.**

What is True and what is False in Python

In Python, the following values are considered False:

- **Zero (Zero may be 0 or 0.0 or 0j).**
- **None (A None evaluates to bool False)**
- **False (A False obviously evaluates to a bool False).**
- **All empty sequences. (Empty lists, such as [], empty tuples, such as () etc. all evaluate to False. But a list containing any element, even a zero, will not evaluate to bool False.**
- **An empty mapping like an empty dictionary, that is, {} will also evaluate to bool False.**

Note that anything except what is mentioned above will evaluate to bool True. So an object of any type always evaluates to bool True.

chr(x) and ord(x)

The function chr(x) does the following:

- **It takes an integer x as a parameter.T**
- **It returns the corresponding "character" to the given integer as parameter.**
- **The integer given to the chr(x) function is treated as a unicode point and the value returned by chr(x) is the corresponding "character".**
- **So you can think of the chr(x) as a "mapping" between a unicode integer number and its corresponding character.**
- **For instance, chr(97) returns the string 'a'.**
- **The valid range for the argument x (To be given to the chr(x) function) is from 0 through 1,114,111 (0x10FFFF in base 16).**

If it is outside this specified range, then ValueError will be raised

Note: The reverse of chr() function is the ord() function. If the string equivalent of a Unicode character is given to the ord() function, then it will return the integer representing the Unicode code. For instance, ord('a') returns 97 while ord('A') returns 65.

cmp(x, y)

The cmp(x,y) function does the following:

- **Compares two objects x and y and returns an integer according to the outcome.**
- **If x < y, the return is -1.**
- **If x == y, return is 0.**
- **If x > y, the return is +1.**

NOTE:- The cmp() method has been deprecated in Python 3.x.

- **So, instead of using cmp(x,y) it is better to use ((x > y) – (x < y).**
- **Note that x > y when joined to x < y with a minus sign ie ‘-‘ then they will be implicitly cast into ints.**
- **So this is equivalent to (int(x > y) – int(x < y)). Further note that bool value True will cast into int 1 and bool value False will cast into int 0.**

divmod(x,y)

The divmod(x, y) function does the following:

- Takes as arguments two numbers x and y.
- It returns a tuple of numbers (q, r), where q is the quotient and r is the remainder.
- If the two arguments x and y are integers, then the result is the same as (x//y, x% y).
- If either of x or y are floats, then q is the whole part of the quotient and r is x –(q*y).
- If y =0, you get Zero Division Error.
- If x = 0, you get (0, 0)

float(x)

The important points regarding this function are as follows:

- The function casts a variable to a floating point number.
- The function takes as parameters either an integer, long or string.
- If the input is a string, it must contain only digits with or without a decimal point and with or without a sign, that is, ‘+’ or ‘-‘.
- The parameter given to the function can be exponential form also, such as 1.0e6 or 1.0E6.
- The return type of the function is a float.

id(object)

The id(object) function does the following:

- Gives the "id" of an object.
- The "id" of an integer is unique and constant for this object.
- You can think of the number returned by the id() function as a unique number given to each object.

int(x)

The important features of this function are as follows:

- It casts a variable to an integer.
- It takes 1 parameter (which may be long, string or float).
- The return type is an integer.
- The function int(x) converts a number or string x to an integer. If no argument is given to the int() function, it returns 0.
- If x cannot be converted to an integer, then an error will be thrown.

len(x)

The len(x) function does the following:

- Returns the length of the given object x.
- The argument x must either be a sequence (such as a string, range, list or tuple.) or a collection (such as a set or a dictionary.)
- If the argument s is a string, then len(s) will return the length of the string s.
- If the argument s is either a sequence or a container, then len(s) gives the number of “items” in the sequence/ container.

range(start, stop[, step])

Some important points to note about the range(start, stop[, step]) are as follows:

- **There are three versions of this function:-**
  - range(stop)
  - range(start, stop)
  - range(start, stop[, step])
- **This function is discussed in detail in the topic on flow control. However, for the present, only one form of the function, that is, range(n) or range(stop) is discussed. Moreover, it is presumed that n is a positive integer. (Other forms of this function with negative integer etc also exist but are discussed later).**
- **range(n) will generate a sequence of numbers from 0 to n-1.**

Here again, there is a difference between Python 2.x and Python 3.x. In Python 2.x on IDLE if you input range(5), output is a list, that is, [0,1,2,3,4]. But in Python 3.x, the output is not a list. Rather, it is a "range object" which is iterable, that is, which can be moved over one by one and can also be converted into other Python objects, such as a list.

round(number[, ndigits])

The function round(number [, ndigits]) does the following:-

- **Takes two parameters and returns the “rounded off value” of the given “number”. The first parameter is “number” and denotes the number, which is to be “rounded off”. The second parameter “ndigits” is optional and denotes the number of digits to which the rounding-off is to be done.**
- **If ndigits is omitted, it defaults to zero.**
- **Again, note that square brackets indicates parameters with default values, and hence, are optional. Here, n digits is in square brackets and it has a default value of 0. So if no value of ndigits is given, it defaults to 0.**

str(object='')

The function str(object= ‘’) does the following:

- Returns a string version of object.
- If an object does not provide “its own string version” then the function str(object), returns the empty string.
- If no argument is given an empty string i.e. (‘’) is created.

You can think of the str() function in two different ways:

First, you can think of it as a “casting” function which creates a cast of the object into a string.

A second way of thinking of the str() function is that it creates a “string representation” of an object. This topic is covered later.

tuple([iterable])

The function tuple([iterable]) does the following:

- Takes an “iterable” as an argument. For the current context, think of an iterable as a sequence or a container.
- The function tuple([iterable]), converts the iterable into a tuple and returns it. Remember that a tuple is a sequence so the “order of items” is important. So when an iterable is converted into a tuple by the tuple([iterable]) function, the order of items in the tuple is the same as in the iterable.
- If the iterable given is already a tuple, then it is returned unchanged.
- Hence, you can give a string, a list or even another tuple to the tuple function.
- If no argument is given, an empty tuple is created.

Module math

(It has a number of very useful functions. To use them you need to import math)

- **math.ceil(x): It returns the "smallest integer not less than x".**
- **math.fabs(x): gives the absolute value of x.**
- **math.floor(x): It returns the "largest integer not greater than x". So you can say that it gives the "floor" of x.**
- **math.exp(x): The function exp(x) returns e**x.**
- **math.log10(x): This method gives the base 10 logarithm of x. It is better to use this method rather than log(x, 10).**
- **math.pow(x, y): This function takes two arguments x and y, where x is the "base" and y is the "power" to which it is raised. The method gives x raised to power y. So, this method is similar to x ** y.**
- **There are many other methods like math.sqrt(x), math.radians(x) etc. You can refer to the online documentation at:**
- **https://docs.python.org/3/library/math.html**

Module random

Why is it called "pseudo random number generator"?

- **The algorithm used to implement the random module in Python is "deterministic" and is based on the initial "seed" given to the function.**
- **What does "deterministic" mean?**
- **It means that given an initial "seed" the same number is generated every time.**
- **This is in contrast to a truly random number, such as when a dice is rolled.**
- **You cannot predict with 100 per cent surety the outcome of a roll of a dice.**
- **Since the algorithm used to generate a random number in the random module is "deterministic", that is, it depends on the "initial seed" value, it is called "pseudo-random number generator".**

random.random()

Important points regarding random.random():

- **This is the basic method of the random module.**
- **It generates a random number in semi-open range [0.0, 1.0).**
- **In the range [0.0, 1), there is a "square bracket" to the left and a "round parenthesis" to the right. The square bracket indicates a "closed interval to the left" and the round parenthesis indicates an "open interval to the right". In other words, 0.0 is possible, 1.0 is not possible. You can say that the number generated is $0 \le x < 1$. Semiopen range here means that the lower limit, that is, 0.0 is included but the upper limit, that is, 1.0 is excluded.**
- **Suppose you want a bigger random number, you can multiply the return value with an appropriate number.**

random.seed([a = None])

Important points regarding random.seed([a = None]):

- The method takes an optional parameter “a” with a default value of None. So you may omit “a”, or if you give a seed of None, then by default, the method uses the current system time as seed value.
- You can give an object as value to the optional parameter “a”.
- However, if you choose to give some object as parameter to the seed() method, then this object must be a “hashable” object.
- So anything, which is immutable can be used as an argument to random([a]), where “a” is an immutable object.
- Since integers, floats, strings, and so on are immutable, they can be used as arguments to random.seed() function.)

random.randint(x, y)

Important points regarding random.randint(x, y):

- **It gives an integer n in the range x ≤ n ≤ y.**
- **Note that randint function can return the upper limit "y" also.**
- **This is unlike other functions in Python where the upper limit is generally excluded, while the other functions always return values smaller than the upper limit.**

random.choice(sequence)

- Here, sequence can be any sequence.
- So sequence could be a string, list or tuple.
- This method returns an item randomly selected from the given sequence.

random.uniform(x, y)

- x and y must be integers
- This method gives a random floating point number f such that x ≤ f ≤ y for x ≤ y. If x ≥ y, then you get x ≥ f ≥ y.
- The upper limit, that is, "y" may or may not be included in the range. Whether "y" will be included or not depends on the rounding in the equation:- x + (y - x) * random().

The use of the equation x + (y - x) * random() will become clear from an example:-

- Suppose you take range (1,2), that is, random.uniform(1,2).
- Further, suppose that the random.random() generates two random numbers, say 0.9999999999999999 and 0.99999999999999999999.
- Now you can see how the equation x + (y - x) * random() behave for these two random numbers on IDLE:-

```
>>>1 + (2-1) * 0.9999999999999999  # Will round off to 1.9999....
1.9999999999999991
>>>1 + (2-1) * 0.99999999999999999999  # Will round off to 2.0
2.0
```

For details see: https://docs.python.org/3/library/random.html#random.uniform

randrange([start], stop, [step])

Important points regarding random.randrange([start], stop, [step]) function are:

- **The function takes three parameters out of which two, namely “start” and “stop” are optional and only one, that is, “stop” is mandatory.**
- **The “start” parameter is the start point of the range. This would be included in the range.**
- **The “stop” parameter is the stop point of the range. This would be excluded from the range.**
- **The “step” parameter indicates the steps to be added in a number to decide a random number.**
- **The function returns a randomly-selected element from range(start, stop, step).**

Syntax for writing / defining your function

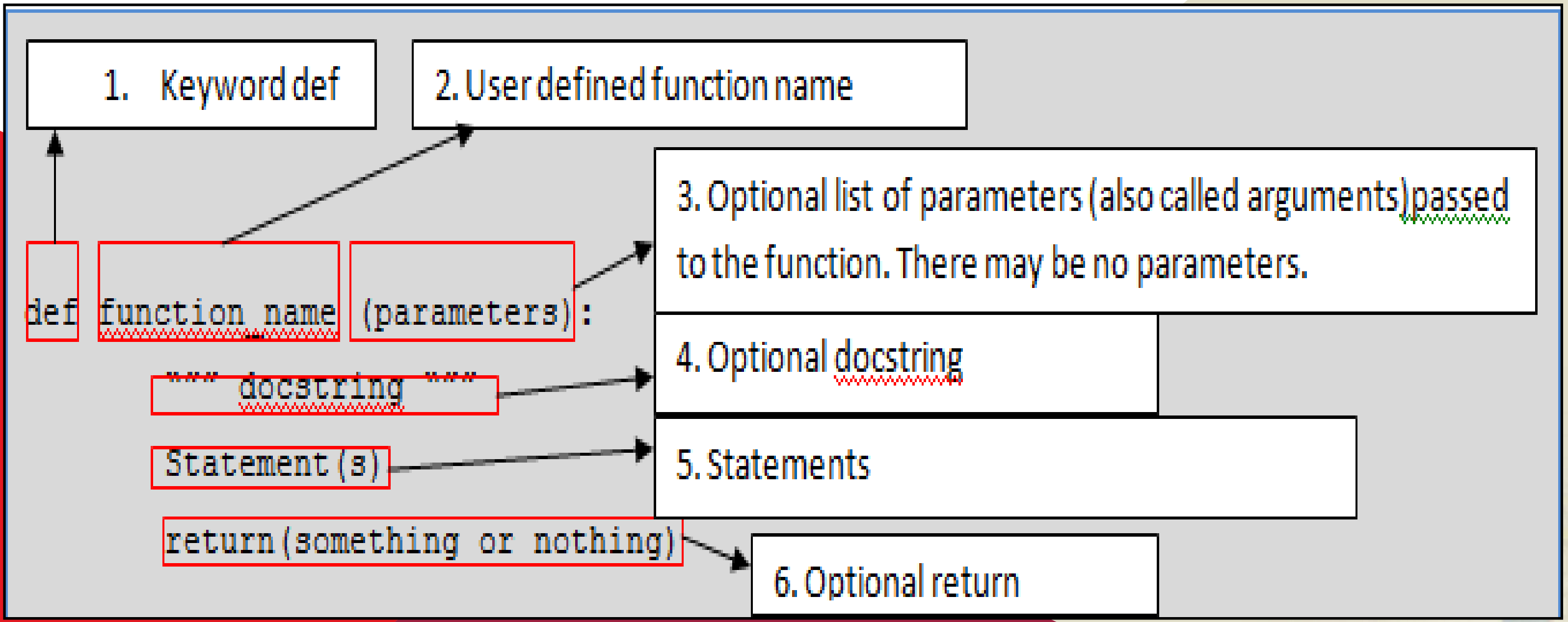
A function definition must have:

- **Keyword "def": This indicates the start of the function header. Defining a function does not call the function. To call the function, you have to make an explicit function call.**
- **A function name: One needs a "function name" to uniquely identify it. The rules for writing a "function name" are the same as that for an "identifier" in Python.**
- **Parameters: After the function name and inside the brackets, there is an optional list of parameters, separated by commas. The term "optional" means that it is possible to have a function with or without any parameter.**
- **A colon (:) is used to mark the end of the function header.**
- **The body of the function has at least one or more valid Python statements. All these statements must have the same indentation level (four spaces).**
- **It also has an optional return statement to return a value from the function.**

Some important points regarding the "return" statement in a Python function/ method

- A "return" statement "terminates" the function call. The return statement is used to exit a function and go back to the place from where it was called.
- It also "returns" the result (If any). The "value returned" by a "return" statement is the "value of the expression" following the return statement.
- If the return statement is without an expression, the special value None is returned.
- So, if you specifically define the function to return an object, it will return that object, but if you don't specify an object, it will return None.
- A "return" statement can be situated anywhere in the function body.
- The body of a function can contain one or more return statements.
- Note that if there is no return statement in the function code, the function ends, when the control flow reaches the end of the function body and the value None will be returned.(In Python, a function will always return something.)

Figure shows important “components” of a function definition


1. Keyword def
2. User defined function name
3. Optional list of parameters (also called arguments) passed to the function. There may be no parameters.
4. Optional docstring
5. Statements
6. Optional return
def
function_name
(parameters):
""" docstring """
Statement(s)
return(something or nothing)

Flow of execution of a function call

- **The “order or sequence” in which statements in a script are executed, is called the flow of execution.**
- **The execution begins with the first statement of the program. Statements are executed one after another from top to bottom.**
- **However, function definitions do not affect the flow of execution of the program. So, this means that as a script is progressing from top to bottom, if it encounters a function definition, it will ignore it and move to the line after the function definition. However, if a function is called, the “flow of execution” will jump to the function definition and then come back to the point from where the function was called.**
- **Note that you can define one function inside another (Also called nested functions). For such “nested” functions, the inner function is executed only after the outer function is called.**
- **So function calls are like a “detour” in the flow of execution. When a script encounters a “function call”, it does not go to the next statement, but rather the program flow jumps to the first line of the called function. It then executes all the statements there. Thereafter, it comes back to pick up from where it had left.**

Local versus global variable (A very simple explanation)

There is more to local/ global variables. It is discussed in the section on “scope” of a variable. For the moment, just accept the following:

- A variable defined inside a function definition, that is, code block beginning with the def statement is “local” to that function.
- A variable in a script, which is outside any def call will be global to that program. This means that when you are inside a function call, the local variable and the global variable both will exist but when you are outside the function call, the local variable ceases to exist and only the global variables exist.

Some point regarding “scope” -1

- What do “names” or “tags” mean in Python? In Python, “names” or “tags” live in what is called a “namespace”.
- In Python, each namespace has a “scope”. The “scope” of a name and a namespace determines its “visibility” or “availability”.
- In Python, a name is given a “namespace” or a “scope” when it is first assigned.
- In some programming languages (Like C++), you can ‘define’ a variable name without giving it a value or an object. You cannot do so in Python.
- In Python, the namespace and scope of an object (represented by a “variable name” or a “tag”) is determined when it is first assigned (Usually using the assignment ie ‘=’ operator).

Some point regarding “scope” -2

- So as pointed out earlier, if a variable is assigned a value inside a function definition, then its name space and scope will be within the function definition only. It will cease to exist once you go out of the function definition. Further, if there is the same variable name, it can exist without clashing provided it is in a different scope.
- Note that the same variable name can be used in different scopes, that is, different name spaces. In local scope, the local name will prevail over the global name and vice-versa.
- The scope of a variable is that part of the script where the variable is “recognized”. A variable, which is defined inside a function will not be visible from outside the function. Hence, such variables have a local scope.
- The lifetime of a variable is the period or “part of script” during which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes. On exiting from a function call, all the local variables inside the function automatically get destroyed.
- Therefore, if a function is called a second time, it will not “remember” the value of a variable from its previous calls.

Because learning changes everything.®

Thank You!

For any queries or feedback contact us at:

support.india@mheducation.com

1800-103-5875

www.mheducation.co.in