

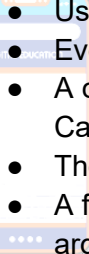
Functions In Python

Functions

Hello, learners. Welcome to the yet another module on modular programming concept using the user defined functions. It's admirable that you're determined to master the Python programming language. Functions are of 4 types. But before that in this module, you will learn about how to write user-defined functions, and their advantages, the concept of a function call, and the parameters along with ample examples.

Let's begin with the introduction to user defined functions.

A function is a chunk of code with a function name that the user specifies.

- 
- Users can call a function by its name.
 - Every function has two components: caller and callee.
 - A caller is a function that calls another function; a callee is a function that was called. Callee is writing the chunk of code with the first level indentation.
 - The code inside a function only runs or executes when it is called.
 - A function can accept data from the caller program, it is called as function parameters or argument.
 - The function parameters are represented inside parentheses and separated by a comma. A function can accept any number of arguments.
 - A function can return data to the caller program. Unlike other popular programming languages, Python function definition doesn't specify the return type. Also it returns multiple return values.
 - You can not use reserved keywords as the function name. A function name must follow the Python Identifiers rules.

Now let us look at the advantages of the functions.

The following are the advantages of Python functions.

- By using functions, we can avoid rewriting the same logic/code again and again in a program.
- You can call python functions any number of times in a program and from any place in a program.
- You can track a large python program easily when it is divided into multiple functions.
- Reusability is the main advantage of python functions.
- However, Function calling is always overhead in a python program.

This brings us to the topic of creating functions.

- You can use def keyword to define the function.
- The function block is started with the colon (:) and all the same level block statements remain at the same indentation.
- A function can accept any number of parameters that must be the same in the definition and function calling.

Now, It's important to understand how to call functions.

- Function must be defined before the function call otherwise, the python interpreter gives an error.
- Once the function is defined, we can call it from another function or from the python prompt.
- To call the function, use the function name followed by the parentheses.

Example

```
def hello_world(): #Function Definition
    print("hello world")
hello_world() #Function Call
```

The output will be hello world.

It is crucial to remember how parameters are assigned in functions.

- The information into the functions can be passed as the parameters.
- The parameters are specified in the parentheses. We can give any number of parameters, but we have to separate them with a comma.

Now let us understand better with these examples.

#defining the function

```
def func (name):
    print("Hi ",name)
```

#calling the function

```
func("Friend")
```

Here the function is defined as 'func' along with the 'name' as a parameter in the parentheses. When you call the function, it prints the output as 'hi friend'.

In the next example.

#Sum of two numbers

```
def sum (a, b):
    print("The sum is",a+b)
```

#calling the function

```
a = int (input("Enter the no 1"))
b = int (input("Enter the no 2"))
sum(a,b)
```

This code of block is written to get the sum of two numbers.

The function is defined as the def sum of a and b. Within the function, it simply prints the value of a+b.

And the sum of a and b is called.

If the value of a and b is taken as user input, we can use the function `sum(a,b)`. Assuming the user inputs to be 110 and 220 respectively, then the sum output is 330.

Please note that following the right coding indentation for function is crucial, otherwise python will prompt error.

Look at the anatomy of the function with return value.

The name in function name - it is an identifier by which the function is called.

In the parentheses is the argument which contains the list of values passed as function.

Four spaces are imperative to add to ensure the indentation in the function body to avoid error.

The function body contains the statements which are executed each time when the function is called.

The return value ends the function call & sends data back to the program.

In this example

Def is the function header followed by the function name which is `add_numbers` and x and y in parentheses are the arguments to be passed as functions.

A docstring is used to describe the code followed by the function body containing statements to execute the function.

In the end, is the return value to get the output.

Let us look at a few more examples to reinforce the concept of functions.

Here is the program to define the function.

```
def odd_even_checker(i):
```

```
    if i % 2 == 0:
        return 'Even'
    else:
        return 'Odd'
```

Now in the first example, `print(odd_even_checker(20))` the function is called to check if the number is odd or even.

Since the modulus operator leaves zero reminders when divided by 20, therefore the output is even.

Similarly, the next function is to check whether the number is odd or even, since the modulus operator leaves one reminder when divided by 15, therefore the output is odd.

I hope the topic of functions is clear to you and we shall proceed to the next topic which is called by reference.

So what is called by reference?

Call by reference it will refer to the memory space rather than the value.

- All the changes made to the reference inside the function revert back to the original value referred to by the reference.
- However, there is an exception in the case of immutable objects since the changes made to the immutable objects like string, tuple, frozenset do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

Here are examples to understand call by reference.

```
#defining the function
def change_list(list1):
    list1.append(20)
    list1.append(30)
    print("list inside function = ",list1)
```

Hereby using the append function, we are adding 20 and 30 to the list.

```
#defining the list
list1 = [10,60,40,50]
```

```
#calling the function
change_list(list1)
print("list outside function = ",list1)
```

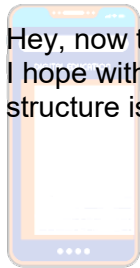
Now we will define the list with the numbers.

Since we have added 20 and 30 by using the append function, the output will show it in the list but in the end.

This change will be reflected in the call function when we call the function with change_list with a list outside the function. The output will be 10, 60, 40, 50, 20, 30.

Hey, now the introduction to functions comes to an end here.

I hope with plenty of examples, the concept of functions, ways to call functions, and their structure is clear to you. See you in the next module.



Types of Python Arguments

Welcome back to the module on types of argument functions.

Let us look at how to define function arguments.

Function definition contains the function name followed by formal arguments in parentheses.

Function call contains the actual parameters or arguments. So the actual values of the arguments are referred to here when a function is called.

This brings us to the first type of argument called keyword arguments.

Keyword arguments of the form can also be used to call functions.

Values given through arguments during a function call do not have to be in the same order as the parameters in the function specification. Keyword arguments can help with this.

However, all keyword arguments must match the function definition's parameters.

Look at the syntax, in the keyword function, the argument is equal to value.

1. The same syntax is applicable both for function definition and function call.
2. If it exist in function definition it is called as default arguments.
3. If it exist in function call it is called as keyword arguments.

You assign values to arguments.

For example,

#Function Definition

```
def add(a,b=5,c=10):  
    return (a+b+c)
```

#Function Call Method 1

```
print (add(b=100,c=15,a=20))
```

#Output:135

In method 1, there's no need to keep the parameters in the same order because they're all keyword arguments.

#Function Definition

```
def add(a,b=5,c=10):  
    return (a+b+c)
```

#Function Call Method 2

```
print (add(a=100))
```

#Output:115

In method 2, Only passing a mandatory argument (a) during a function call. The arguments b and c are default arguments.

Moving on to the second type of argument which is called positional arguments.

Values given through arguments in a function call should be in the order of parameters in the function declaration. Positional arguments are what they're called.

The positional arguments can be passed in two ways. Let us discuss the methods with the help of examples.

#Function Definition

```
def add(a,b,c):  
    return (a+b+c)
```

The function has been defined.

```
#Function Call Method 1
print (add(100,20,30))
```

In method 1, all arguments are given as positional arguments during the function call. Argument values are given to parameters according to their position. A is given the number 100, b is given the number 20, and c is given the number 30. Therefore the output of the sum is 150.

Similarly in the second method.

```
#Function Definition
def add(a,b,c):
    return (a+b+c)
#Function Call Method 2
print (add(100,c=30,b=20))
#Output:150
```

In method 2 always remember, if you're using a combination of positional and keyword arguments, the keyword arguments should always come after the positional arguments.

Let us discuss the third type of argument called default argument.

Default arguments are values that are provided when functions are defined.

The assignment operator = is used to assign a default value to the argument.

- During function calls, default arguments become optional.
- The default value is overridden or modified if a value is passed to the default parameters during function calls.
- Any number of default arguments can be passed to the function.
- Non-default arguments should be given precedence over default arguments.

Look at the examples to understand various methods.

```
#Function Definition
def add(a,b=5,c=100):
    return (a+b+c)
```

```
#Function Call Method 1
print(add(3))
#Output:108
```

Here in method 1, the Function call can be used by providing only the mandatory argument.

In this case, a will by default will take the value of 3 and accordingly will give the output 108.

```
#Function Definition
def add(a,b=5,c=100):
```

```
    return (a+b+c)
The function is defined.
#Function Call Method 2
print(add(10,20))
#Output:130
```

In method 2, a function call can be used by providing one of the optional arguments.
Here the value of a and b have been reassigned with 10 and 20 as optional arguments.

```
#Function Definition
def add(a,b=5,c=100):
    return (a+b+c)
#Function Call Method 3
print(add(20, 30, 40))
#Output:90
```

In method 3, the Function call is used by providing all the arguments.

The values of the arguments a, b, and c are reassigned as 20,30, and 40 respectively by giving the output 90.

I am sure that these three methods of default arguments are easy to understand.

We will discuss the remaining arguments in the next module, stay tuned.

Variable Length or Arbitrary Arguments

Welcome to the module on functions, we have covered the two types of arguments called keyword and default arguments.

In this module, you will learn about Variable Length or Arbitrary Arguments and their types.

Let's get started with Variable Length or Arbitrary Arguments.

Arbitrary arguments are sometimes known as variable-length arguments. If we don't know how many arguments the function will require in advance, we can use arbitrary arguments.

There are two types of arbitrary arguments.

1. Arbitrary positional arguments
2. Arbitrary keyword arguments

Let us understand Arbitrary Positional Arguments.

An asterisk (*) is used before a parameter in a function definition that can hold non-keyword variable-length parameters for arbitrary positional arguments. A tuple will be used to hold these arguments. Here are some examples.

#Sum of n numbers

```
def add(*b):  
    result=0  
    for i in b:  
        result=result+i  
    return result
```

A tuple data structure is created to pass the arbitrary arguments to b.

#Function Call Method 1

```
print (add(10,20,30,40,50))
```

#Output:150

In method 1, the function call is carried out with 5 arguments 10, 20, 30, 40, 50.

And it produces the output 150.

Where as in method 2

#Function Call Method 2

```
print (add(10,20,30))
```

#Output:60

Method 2:

The function call is carried out with 3 arguments 10, 20, 30 and it produces the output 60.

Let us now understand the combination of normal arguments & Arbitrary Positional Arguments.

There may be zero or more normal arguments before the variable number of arguments. Look at an example.

#Sum of n numbers

```
def summat(a, *b):  
    result=a  
    for i in b:  
        result=result+i  
    return result
```

#Function Call

```
print (summat(10,20,30,40,50))
```

#Output:150

The function call is carried out with 5 arguments 10, 20, 30, 40, 50. The first value 10 is assigned to a. The rest of the elements are assigned as a tuple data structure to the arbitrary

argument b.

The sum is 150 produced as the output.

Moving on to the second type of Arbitrary Arguments called Arbitrary Keyword Arguments.

A double asterisk (**) is placed before a parameter in a function that can hold keyword variable-length parameters for arbitrary positional arguments. The syntax - argument is equal to value.

Here is an example of Arbitrary Keyword Arguments

Function to Display Course Details

```
def course(**a):
```

```
    for i in a.items():
```

```
        print (i)
```

#Function Call

```
course(year=2021,course="Python Essential",level="Beginner")
```

Here the keywords are used to define the argument using variable-length parameters. Therefore the output will be.

#Output

```
('year', 2021)
```

```
('course', 'Python Essential')
```

```
('level', 'Beginner')
```

Keyword variable-length parameters are passed with the syntax argument = value.

We have come to the last topic of the module called Recursive Function.

Recursion is the process of calling a function itself until a termination condition is met.

- One or more base cases can be solved immediately without the need for further recursion.
- With each recursive call, the answer gets closer to the base case.
- The expression evaluation will be carried out using the internal memory called a stack.

Base condition, Iterative expression, and Termination condition must all be considered while writing a recursive function.

Look at the example to understand better. We will get the factorial of the number entered.

```
def factorial(x):
```

```
    """This is a recursive function
```

```
    to find the factorial of an integer"""
```

```
    if x == 1:
```

```
        return 1
```

```
    else:
```

```
        return (x * factorial(x-1))
```

```
num=int(input("Enter the number"))
```

```
print("The factorial of", num, "is", factorial(num))
```

So if the entered number is 5 then the value of x in def factorial is 5.

In this programme, if $x==1$, return 1 is the termination condition.

else :

return (x * factorial(x-1)) is the iteration condition. This is called a recursive expression. The evaluation will happen using internal stack memory..

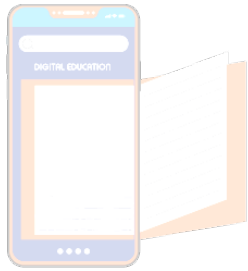
Since 5 is not equal to 1, it will go to the else condition. This 5 will be pushed on to stack data structure or stack memory internally.

The return statement is expanded to 5 multiplied by a factorial of 5-1 which is 4. Subsequently, the function call factorial is executed.

This 5 will be pushed to the stack.

Then the output, the factorial of 5 is 120.

Kudos to you for the successful completion of the important concept called mastering on user defined function.



vityarthi