A Python function is a group of code. To run the code in a function, you must call the function. A function can be called from anywhere after the function is defined. Functions can return a value using a return statement. Functions are a common feature among all programming languages.

There are different types of fucntions in python, some of them are

1. Built-In function
2. Lambda function
3. Recursion function
4. User defined function

**User-defined functions Syntax:**

def function\_name(parameters/arguments):

“String” (if any)

statements/code snippets

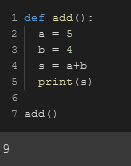
return()

**User-defined functions types:**

1. No argument nothing return
2. Argument but nothing return
3. No argument but return
4. Argument and return

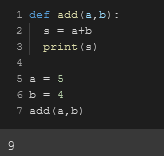
**No argument nothing return**

In a function, there will be no arguments and won’t return anything. Only print within the function after internal calculations. But the function has to be called.

****

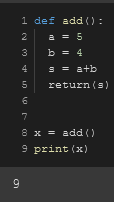
**Argument but nothing return**

In a function, there will be arguments but won’t return anything. Only print within the function after internal calculations. But the function has to be called.

****

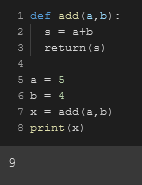
**No argument but return**

In a function, there won’t be any arguments but won’t return anything. Because there are no arguments, there has to be local variables doing the jobs superseding the argument’s role within a function.returns a statement after internal calculations within a function. But the function has to be called.

****

**Argument and return**

In a function, there will be any arguments and a statement will be returned. After executing some operations within a function with the passed arguments, the function returns a statement back to the main code.

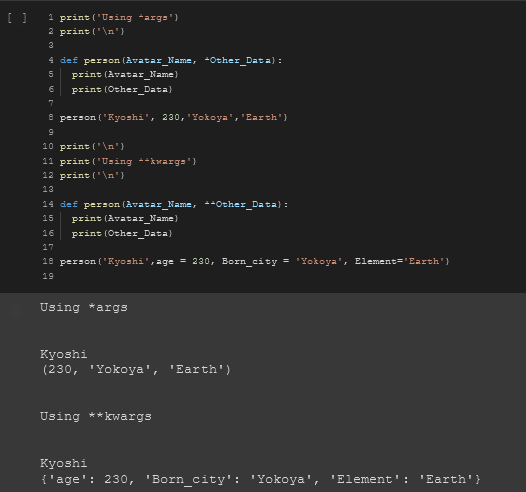
****

**\*args & \*kwargs**

Sometimes, We might not know how many arguments need to be passed in a function. So in that case, We use \*args & \*kwargs. These are special symbols used to pass arguments in a function. We use \*args to pass non-keyword arguments and \*kwargs to pass keyword-arguments.

args is expressed as \*args

Kwargs is expressed as \*\*args

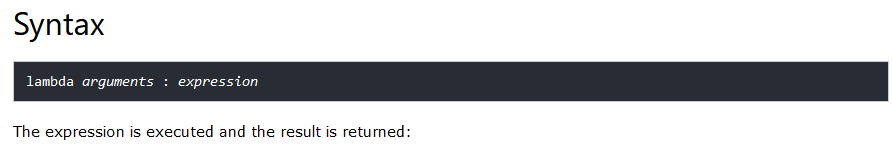


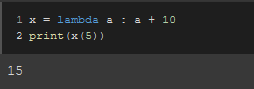
**Lambda Function**

You can write your very own Python functions using the def keyword, function headers, docstrings, and function bodies. However, there's a quicker way to write functions on the fly, and these are called lambda functions because you use the keyword lambda.

Some function definitions are simple enough that they can be converted to a lambda function. By doing this, you write fewer lines of code, which is pretty awesome and will come in handy, especially when you're writing and maintaining big programs.

**Lambda Function:** Here we rewrite our function raise\_to\_power as a lambda function. After the keyword lambda, we specify the names of the arguments; then, we use a colon followed by the expression that specifies what we wish the function to return.

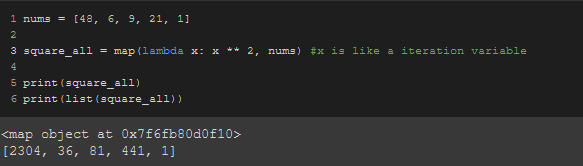




**map() and lambda Function:**

The map function takes two arguments, a function and a sequence such as a list and applies the function over all the elements of the sequence. We can pass lambda function to the map without even naming them, and in this case, we refer to them as anonymous functions.

In this example, we use map() on the lambda function, which squares all elements of the list, and we store the result in square\_all.

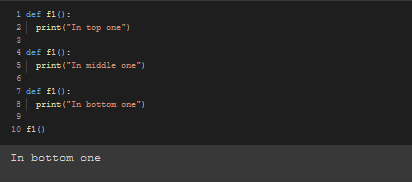


**Miscellaneous**

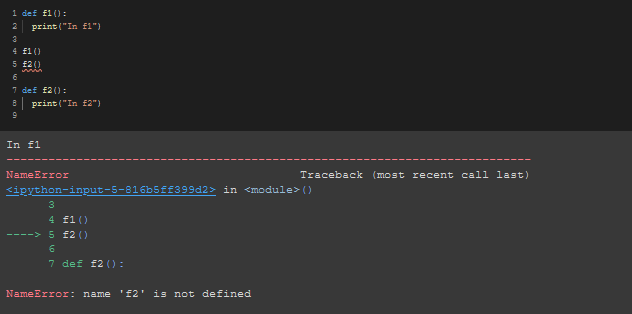
* ***Recursion is a function that calls itself*** which ultimately leads to **“RecursionError”** error.



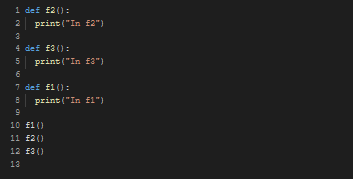
* When a function is called, it only goes up to search for it and will choose the function that’s closest to the body code.



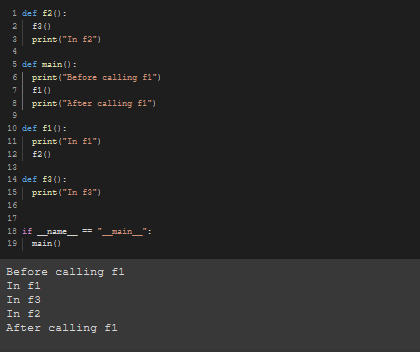
* If a function is defined after the driver code, An **“NameError”** error will occur.



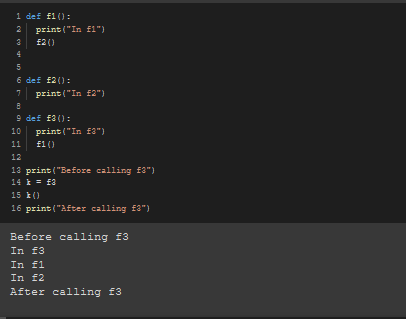
* Sequence/ order doesn't matter as long as it's before the driver/body code



* Function order doesn’t matter if we do show in the picture below.

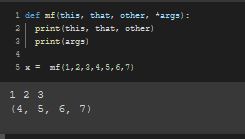


* Function names can be saved in a variable and we can use that variable to call the function.

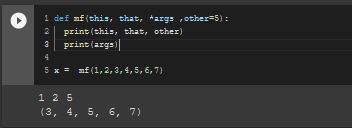


* If you want to print/use all the arguments in a limited parameterized function, Use **\*args** to print/use all the rest of the arguments in the function.

Those arguments goes in the function as a tuple.

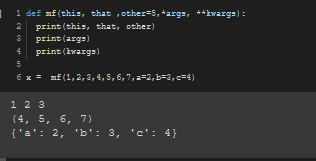


* Order does matter even when using \***args**

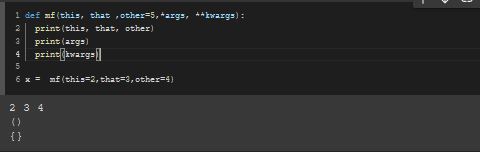


* If you want to print all the arguments in a limited parameterized function and some of the arguments are saved in a variable and you still want to print/use them along with those keyworded variables in the function, Use **\*kwargs** to use all the rest of the arguments in the function.

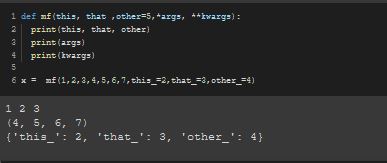
Those keyworded arguments enter the function in the form of a dictionary.



* If nothing’s given as the **\*args** & **\*kwargs** in a function, They remain emply in the form of a empty tuple and dictionary.



* You can set a substitute value as the parameter in the function and that will only work if the value is not set as the arguments when calling the function. Otherwise, It won’t work.



* Python always executes each line from the left side.

In the example below, The value of x[0] is changing as the function got called.

