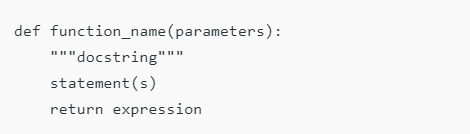
What are Functions?

**Functions** is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

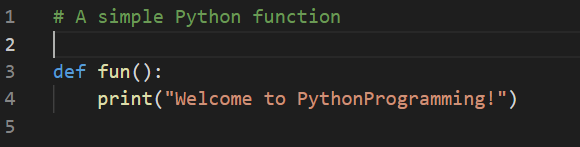
Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

**Syntax:**



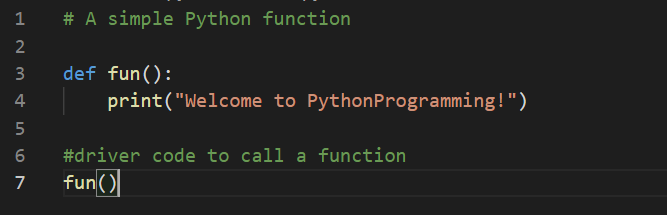
## **Creating a Function**

We can create a Python function using the **def** keyword.



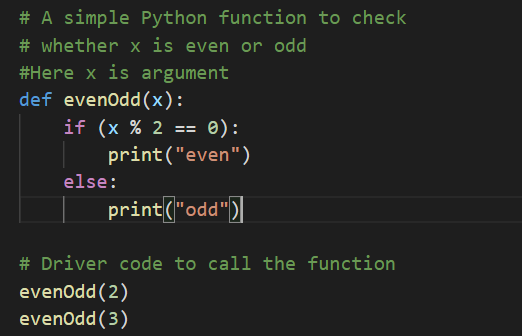
## Calling a Function

After creating a function, we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.



## Arguments of a Function

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

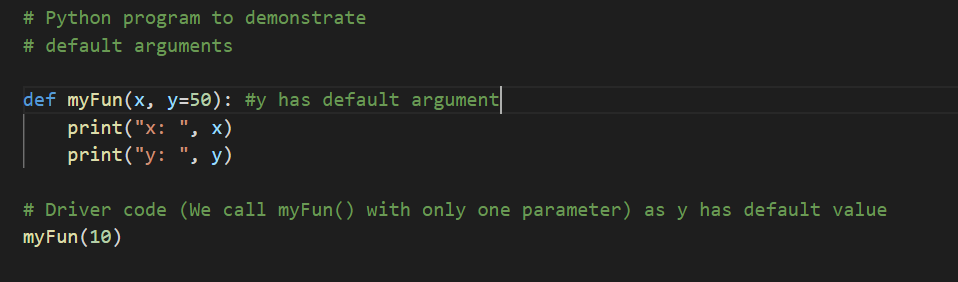


## Types of Arguments

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

### [**Default arguments**](https://www.geeksforgeeks.org/default-arguments-in-python/)

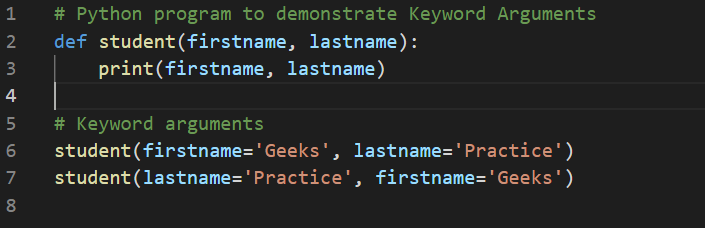
A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.



Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

### **Keyword arguments**

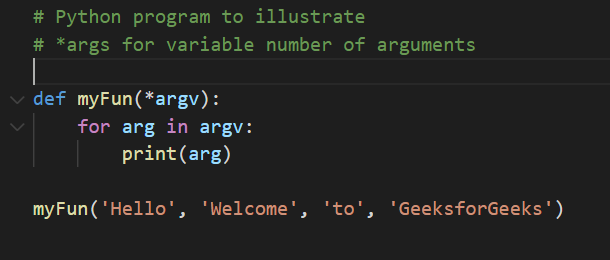
The idea is to allow the caller to specify the argument name with values so that caller does not need to remember the order of parameters.

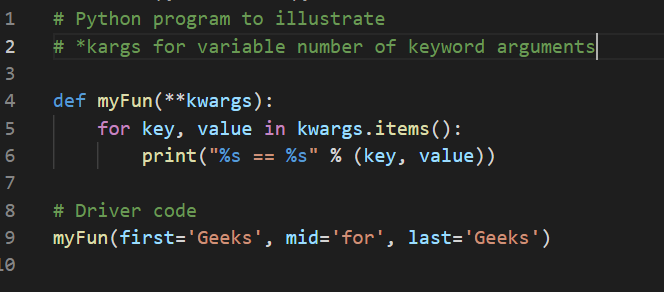


### [Variable-length arguments](https://www.geeksforgeeks.org/args-kwargs-python/)

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

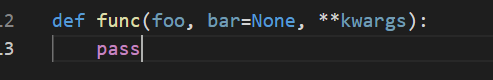
* \*args (Non-Keyword Arguments)
* \*\*kwargs (Keyword Arguments)





# Parameters Vs Arguments

Parameters are defined by the names that appear in a function definition, whereas arguments are the values actually passed to a function when calling it. Parameters define what types of arguments a function can accept. For example, given the function definition:



foo, bar and kwargs are parameters of func. However, when calling func, for example:



the values 42, 314, and somevar are arguments.

The glossary defines them as:

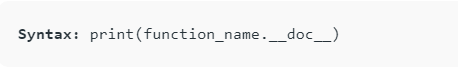
* [**Argument**](https://docs.python.org/2.7/glossary.html#term-argument): A value passed to a function (or method) when calling the function.
* [**Parameter**](https://docs.python.org/2.7/glossary.html#term-parameter): A named entity in a function (or method) definition that specifies an argument (or in some cases, arguments) that the function can accept.

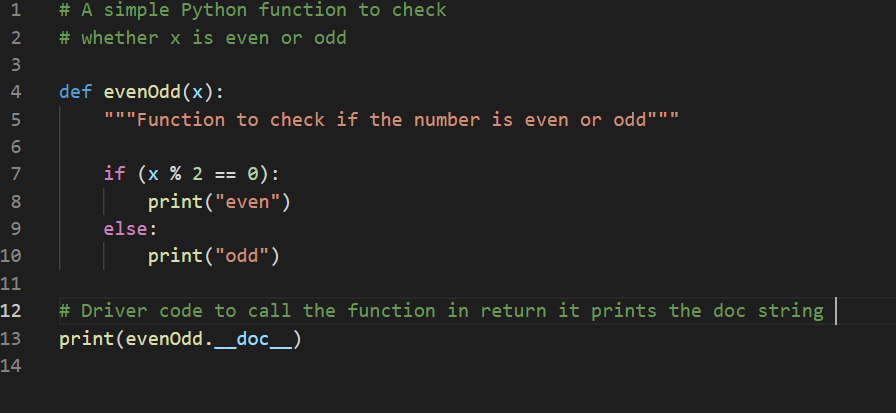
Python doesn't really have "variables" like some other languages - it has "names" referring to "objects".

## Docstring

The first string after the function 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.

The below syntax can be used to print out the docstring of a function:



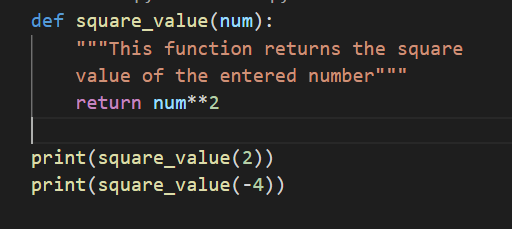


## The return statement.

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

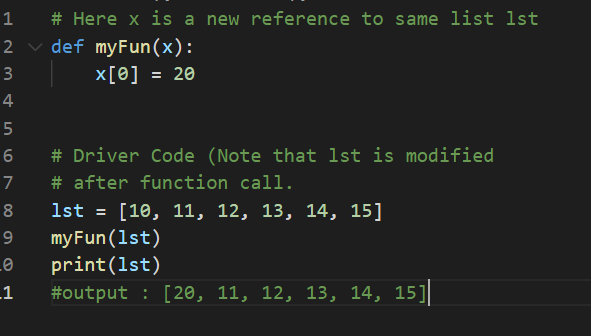


The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

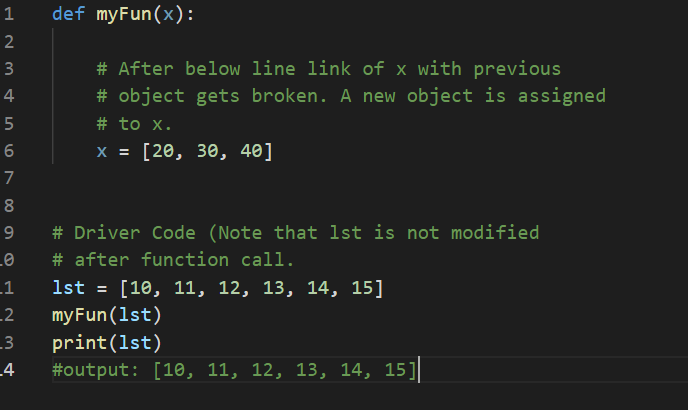


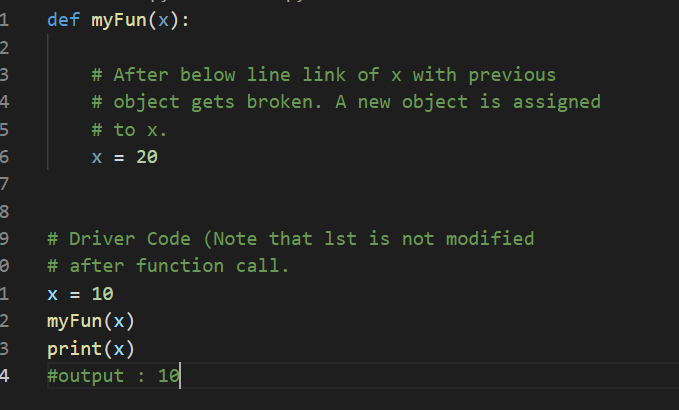
## Is Python Function Pass by Reference or pass by value?

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.



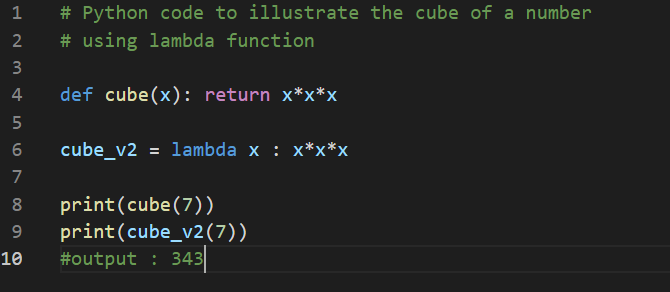
When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken.





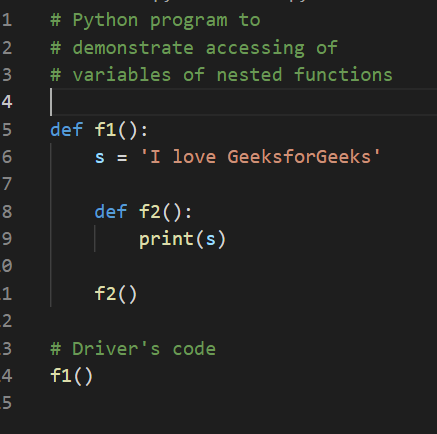
## Anonymous functions:

In Python, an anonymous function means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.



# **Nested Functions:**

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

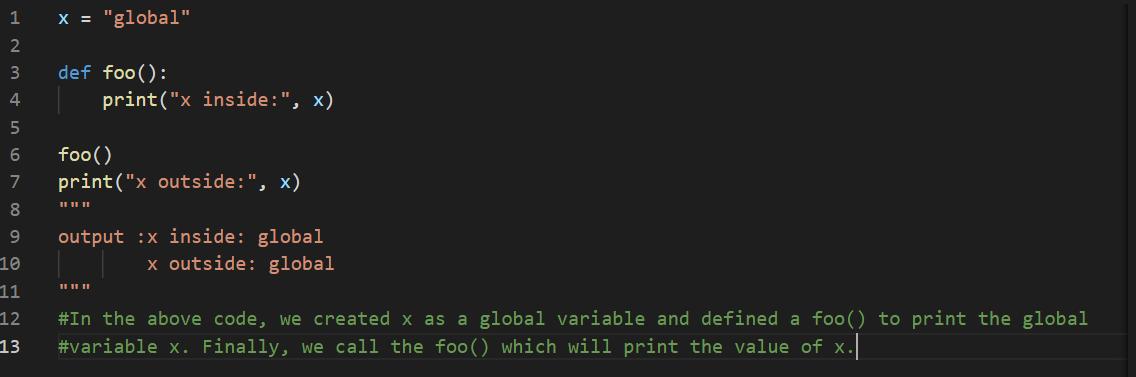


# **Variable Scopes:**

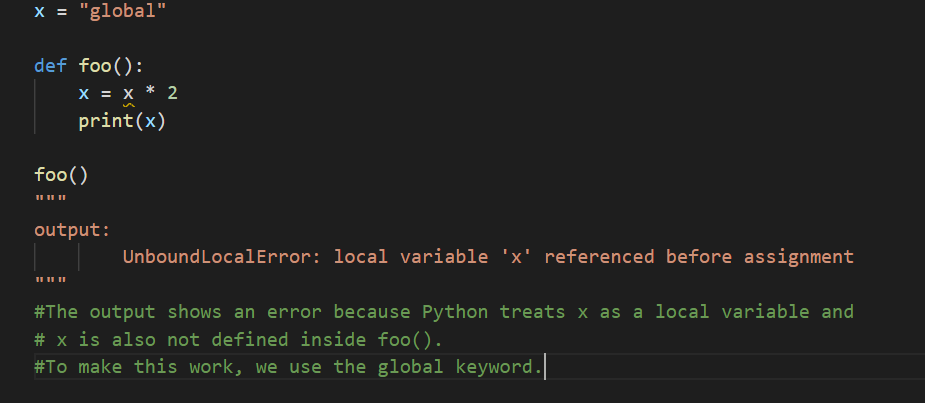
* Global
* Local
* Non-Local

## Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

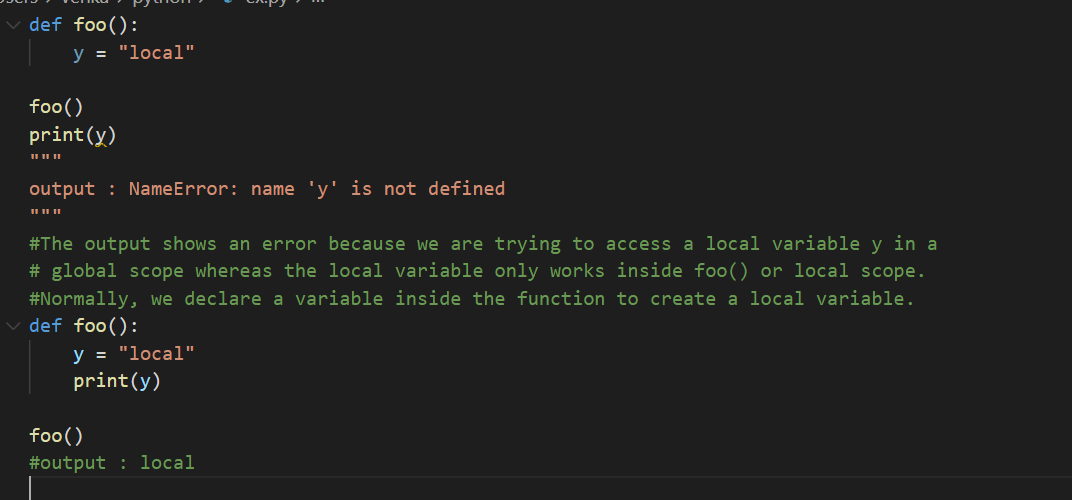


What if you want to change the value of x inside a function?

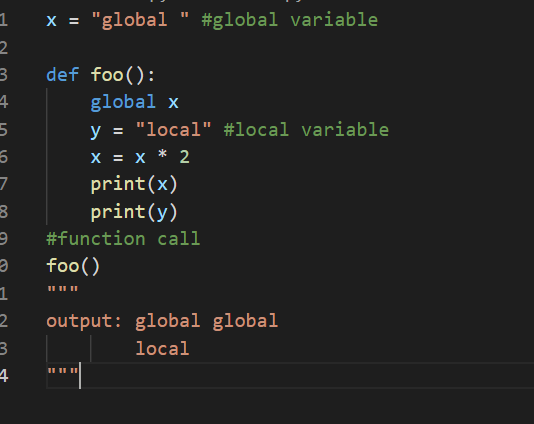


## Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.

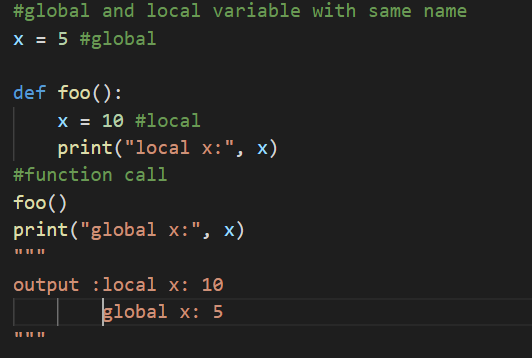


## Global and local variables



In the above code, we declare x as a global and y as a local variable in the foo(). Then, we use multiplication operator \* to modify the global variable x and we print both x and y.

After calling the foo(), the value of x becomes global global because we used the x \* 2 to print two times global. After that, we print the value of local variable y i.e local.



In the above code, we used the same name x for both global variable and local variable. We get a different result when we print the same variable because the variable is declared in both scopes, i.e. the local scope inside foo() and global scope outside foo().

When we print the variable inside foo() it outputs local x: 10. This is called the local scope of the variable.

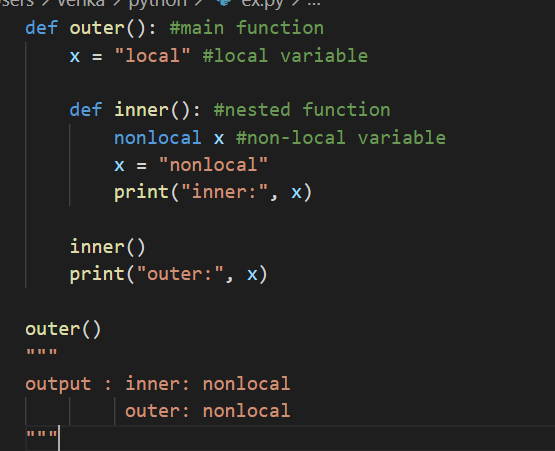
Similarly, when we print the variable outside the foo(), it outputs global x: 5. This is called the global scope of the variable.

## Nonlocal Variables

Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Let's see an example of how a nonlocal variable is used in Python.

We use nonlocal keywords to create nonlocal variables.



In the above code, there is a nested inner() function. We use nonlocal keywords to create a nonlocal variable. The inner() function is defined in the scope of another function outer().

**Note** : If we change the value of a nonlocal variable, the changes appear in the local variable.