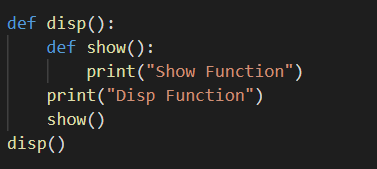
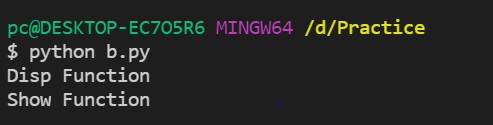
# Nested Function

When we define one function inside another function, it is known as Nested Function or Function Nesting.

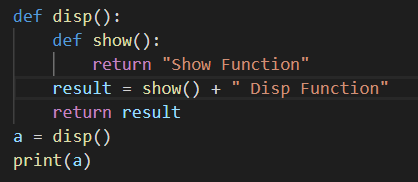
**Example 1:**



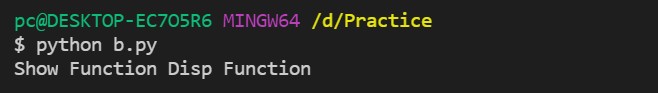
**OUTPUT**



**Example 2:**



**OUTPUT**



**Scope and Namespaces**

**What are Namespaces in Python**

A namespace is a simple system to control the names in a program. It ensures that names are unique

and won’t lead to any conflict. It will allow us to reuse a name in a program.

**Local Namespace**

This namespace covers the local names inside a function. Python creates this namespace for every function called in a program. It remains active until the function returns.

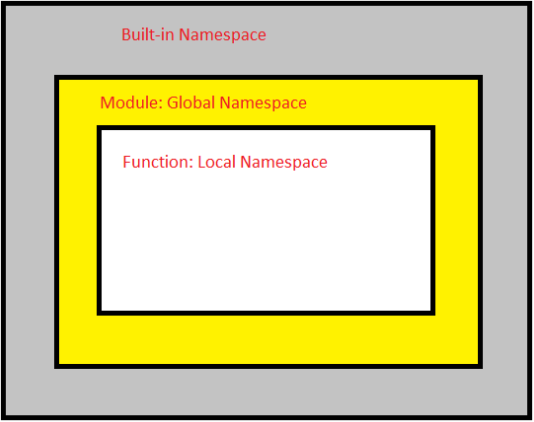
**Global Namespace**

This namespace covers the names from various imported modules used in a project. Python creates this namespace for every module included in your program. It’ll last until the program ends.

**Built-in Namespace**

This namespace covers the built-in functions and built-in exception names. Python creates it as the interpreter starts and keeps it until you exit.

These different namespaces are isolated. Hence, the same name that may exist in different modules do not collide.



Although there are various unique namespaces defined, we may not be able to access all of them from every part of the program. The concept of scope comes into play.

# What is Scope in Python

Namespaces make our programs immune from name conflicts. However, it doesn’t give us a free ride to use a variable name anywhere we want. Python restricts names to be bound by specific rules known as a scope.

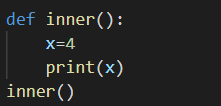
**Note:** Scope refers to the coding area where a particular Python variable is accessible.

# Type of variable Scope LEGB Rule

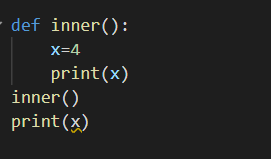
## Local Scope

Contains Names defined inside the current function. If we define any variable inside a function then the scope of that variable is inside that function. When the function ends the availability of that variable also ends.

**Example:**



But if I try to access this variable outside this function, I just try to print x again. Now you can see name error.



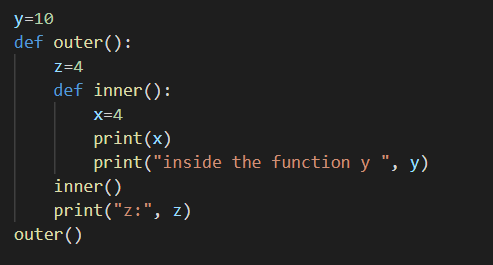
**OUTPUT**



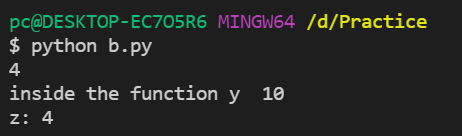
**Enclosed Scope**

Contains Names defined inside any and all enclosed function. It contains names defined inside any and all enclosing function so if we take the nested function. If we define a function inside another function then that is called as nested function. In that case, you see the enclosed scope. Variable declared here is called as enclosed variable.

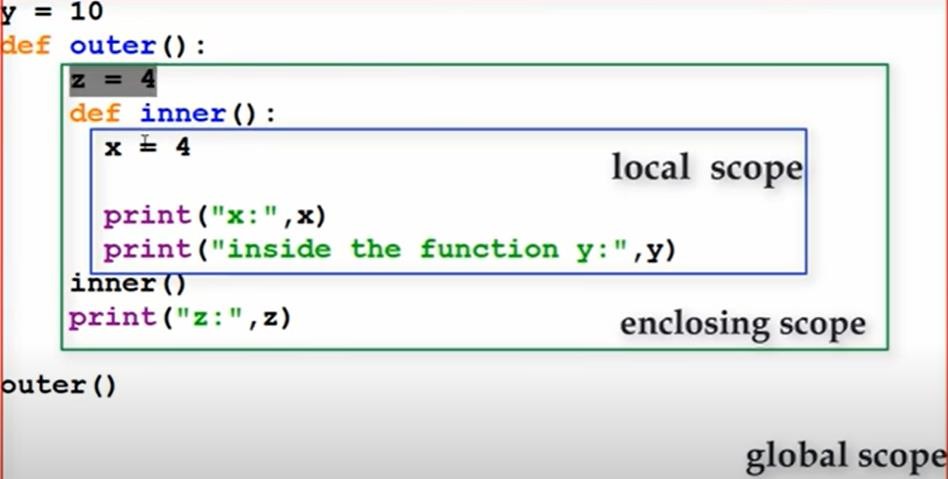
**Example:**



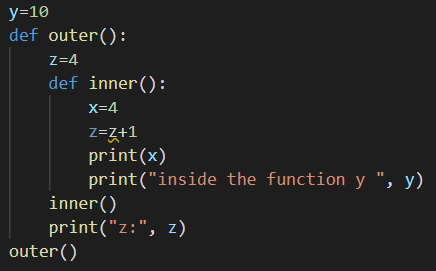
**OUTPUT**



You can see variable **z** is local to the outer function but it is **non local** to inner function, for the inner function **z** is not the local variable as well as it is not the global variable so that’s why it is called as **nonlocal variable** and this is also called as **enclosing variable.**



I am trying to modify the enclosed variable inside the local scope so here we can see the error **unboundlocalerror** so we can’t do that.

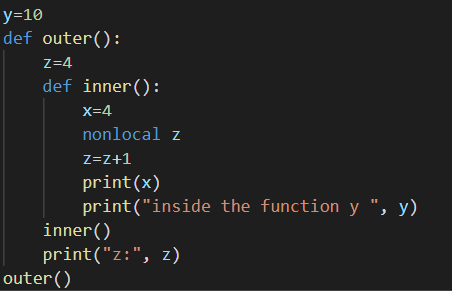


**OUTPUT**

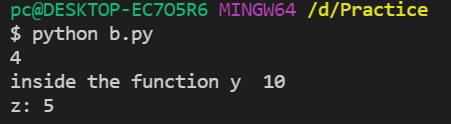


To do this so next similar to the **global** keyword we can use **nonlocal keyword**

to modify the enclosed variable in the local scope.



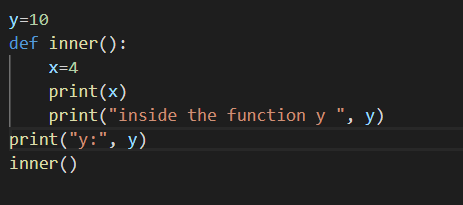
**OUTPUT**



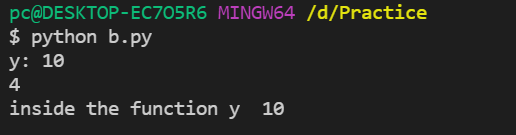
**Global Scope**

Contains Names defined at the top level of the script or module. These names are available for the whole script, the lifetime of the names end when the program ends.

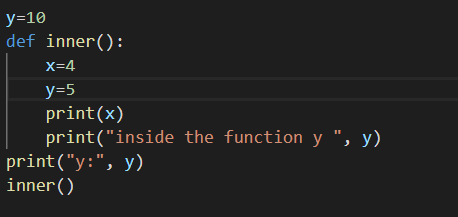
**Example:**



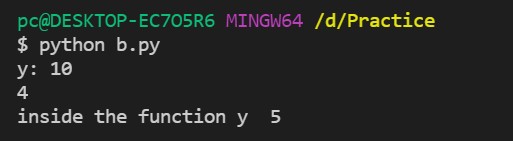
**OUTPUT**



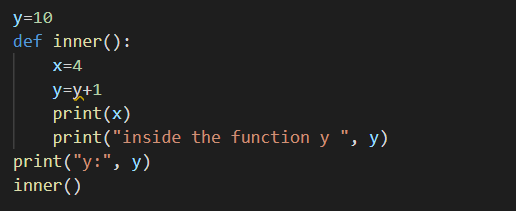
**Example 2:**



**OUTPUT**



Now if I try to modify the global variable in local scope then what will happen……..

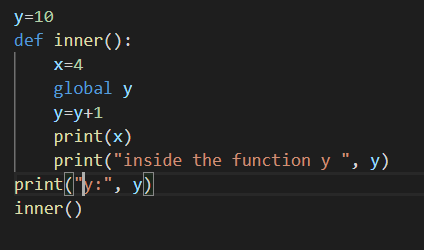


**OUTPUT**



We can’t modify the global variable inside the local scope, if you want to do that then you should use

a keyword that is called as **global.**



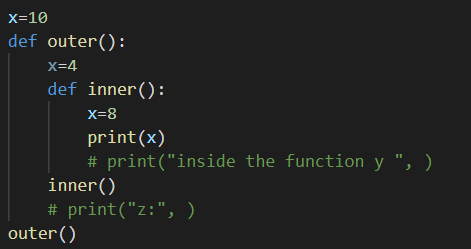
**OUTPUT**



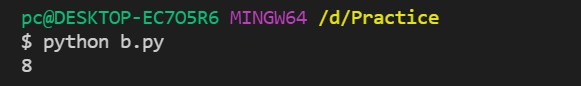
**Built-in Scope**

Contains Names built-in to the python language through the special built-in module ok so it contains built-in names.

But if every scope contains same variable name and if you are using the variable **x** in all the scope then what will be the result. So how python will search for this value, how python will get to know which **x** value it should print, whether it should print **10** now **4 or 8** so if I save this and run, we can see the output that is **8.** This is printing the local value but how python will do this. That is done by using **LEGB** rule. That is nothing but python will first search in the local scope it will search for the name in the local scope if it is not found then it will search in the enclosed scope, if it is not found then it will search in the global scope, if it not found it will search in built-in scope then also it is not found it will give **error**.



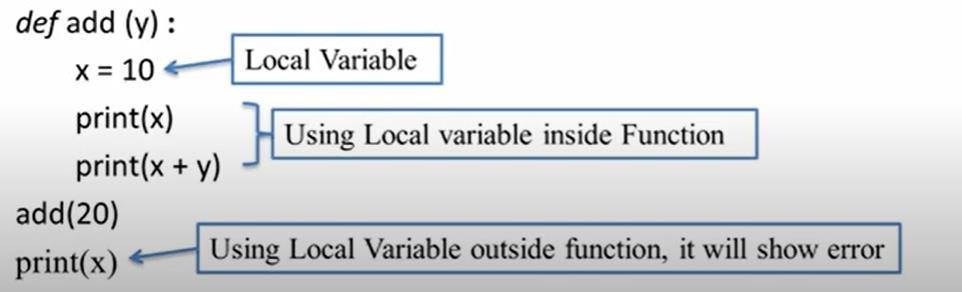
**OUTPUT**



**Global and local and non-local variable Local Variable**

The variable which are declared inside a function called as Local Variable. Local variable scope is limited only to the function where it is created. It means local variable value is only in that function not outside of that function.

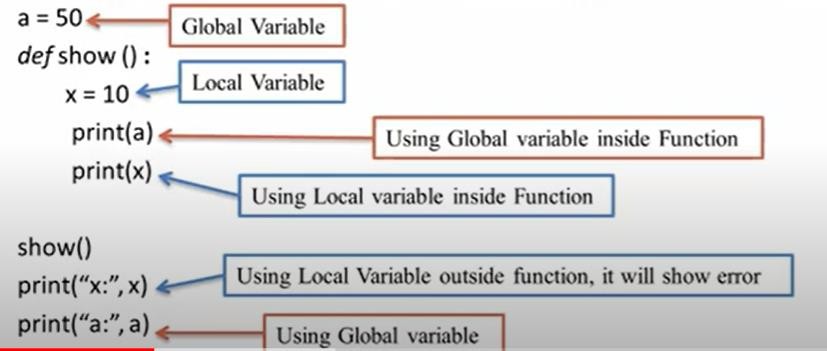
**Example:**



**Global Variable**

When a variable is declared above a function, it becomes global variable. These variables are available to all the function which are written after it.

**Example 1:**

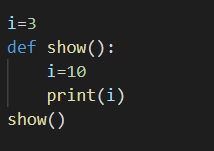


**Global Keyword**

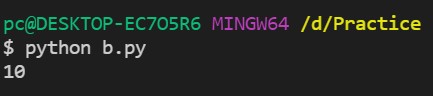
If local variable and global variable has same name then the function by default refers to the local variable and ignores the global variable. It means global variable is not accessible inside the function but possible to access outside of function.

In this situation, if we need to access global variable inside the function, we can access it using **global keyword** followed by variable name.

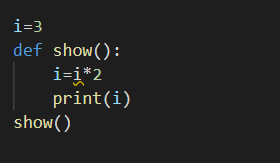
**Example:**



**Output**



**Example 2:**



**OUTPUT**

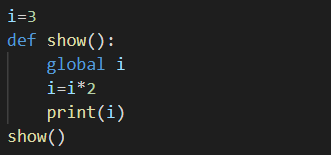


The output shows an error because python treats **i** as a local variable and **x** is not defined inside

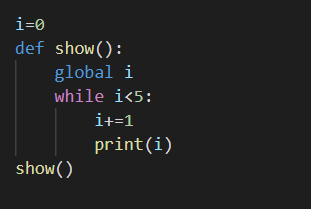
**show().**

To make this work, we use the **global** keyword.

**Solution**



**Example 2:**

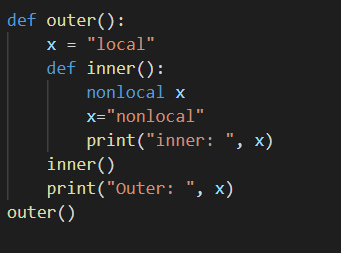


**Non–local Variables**

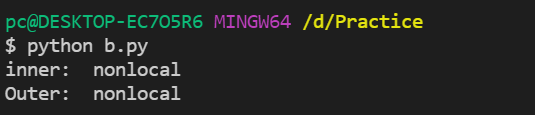
Non-local variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Let’s see an example of how a nonlocal variable is used in python. We use **nonlocal** keyword to create nonlocal variables.

**Example:**



**OUTPUT**



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