# **Python Namespace and Scope**

To simply put it, a namespace is a collection of names.

In Python, we can imagine a namespace as a mapping of every name we have defined to corresponding objects.

It is used to store the values of [variables](https://www.programiz.com/python-programming/variables-constants-literals) and other objects in the program, and to associate them with a specific name.

This allows us to use the same name for different variables or objects in different parts of your code, without causing any conflicts or confusion.

## **Types of Python namespace**

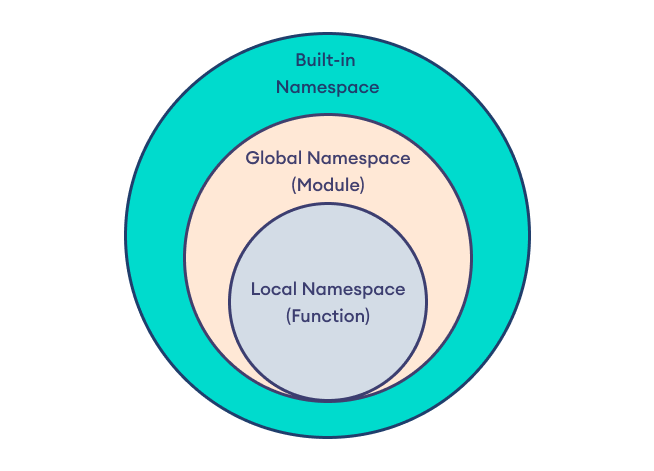
A namespace containing all the built-in names is created when we start the Python interpreter and exists as long as the interpreter runs.

This is the reason that built-in functions like [id()](https://www.programiz.com/python-programming/methods/built-in/id), [print()](https://www.programiz.com/python-programming/methods/built-in/print) etc. are always available to us from any part of the program. Each [module](https://www.programiz.com/python-programming/modules) creates its own global namespace.

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

Modules can have various [functions](https://www.programiz.com/python-programming/function) and [classes](https://www.programiz.com/python-programming/class). A local namespace is created when a function is called, which has all the names defined in it.

Similar is the case with class. The following diagram may help to clarify this concept.

Python Namespaces

## **Python Variable Scope**

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.

A scope is the portion of a program from where a namespace can be accessed directly without any prefix.

At any given moment, there are at least three nested scopes.

1. Scope of the current function which has local names
2. Scope of the module which has global names
3. Outermost scope which has built-in names

When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.

If there is a function inside another function, a new scope is nested inside the local scope.

## **Example 1: Scope and Namespace in Python**

# global\_var is in the global namespace

global\_var = 10

def outer\_function():

# outer\_var is in the local namespace

outer\_var = 20

def inner\_function():

# inner\_var is in the nested local namespace

inner\_var = 30

print(inner\_var)

print(outer\_var)

inner\_function()

# print the value of the global variable

print(global\_var)

# call the outer function and print local and nested local variables

outer\_function()

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Output

10

20

30

In the above example, there are three separate namespaces: the global namespace, the local namespace within the outer function, and the local namespace within the inner function.

Here,

* global\_var - is in the global namespace with value 10
* outer\_val - is in the local namespace of outer\_function() with value 20
* inner\_val - is in the nested local namespace of inner\_function() with value 30

When the code is executed, the global\_var global variable is printed first, followed by the local variable: outer\_var and inner\_var when the outer and inner functions are called.

## **Example 2: Use of global Keyword in Python**

# define global variable

global\_var = 10

def my\_function():

# define local variable

local\_var = 20

# modify global variable value

global global\_var

global\_var = 30

# print global variable value

print(global\_var)

# call the function and modify the global variable

my\_function()

# print the modified value of the global variable

print(global\_var)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Output

10

30

Here, when the function is called, the [global keyword](https://www.programiz.com/python-programming/global-keyword) is used to indicate that global\_var is a global variable, and its value is modified to 30.

So, when the code is executed, global\_var is printed first with a value of 10, then the function is called and the global variable is modified to 30 from the inside of the function.

And finally the modified value of global\_var is printed again.