🡪

命名空间在Python中犹如字典，其中key保存变量名，而value保存着对应的变量。相同的变量名在不同的命名空间中互不影响。

Python中存在四种不同的命名空间：

Built-in(内置命名空间)

Global(全局命名空间) ,使用内置函数globals()用于访问以及修改。Globals()返回的是代表全局变量的引用。当获取引用后，有新的全局变量出现时，使用该引用可以获取。也可以直接对其引用加入全局变量。

Enclosing（封闭式命名空间），一般出现在函数中定义函数。父函数的命名空间为封闭式命名空间。

Local(局部命名空间),使用内置函数locals()用于访问以及修改。与Globals()不同，locals()返回的是代表局部变量的字典的拷贝。所以有新的局部变量出现，前面获取的字典拷贝不会改变。而直接对locals()返回的字典添加值，也不会真的改变局部变量。

当Python程序运行时，Python会在必要时创建命名空间，也会在命名空间不需要时，删除命名空间。

🡪

Built-in(内置命名空间)包含所有Python内置对象。Built-in命名空间在Python程序运行期间一直存在。

可以使用dir(\_\_builtins\_\_)查看

🡪

全局命名空间(Global namespace),全局命名空间包含在主程序中定义的变量。全局命名空间在主程序体开始时创建，直到Python程序结束。

🡪

Enclosing（封闭式命名空间）以及Local(局部命名空间)。

当方法执行时，Python将创建一个局部命名空间直到该方法结束。

例如：

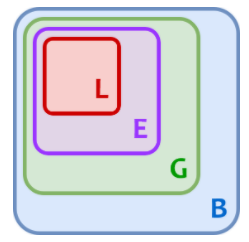
def f(): # f()函数执行时创建封闭式命名空间  
 print('Start f()')  
  
 def g(): # g()函数执行时创建本地命名空间  
 print('Start g()')  
 print('End g()')  
 return  
 g()  
 print('End f()')  
 return  
  
  
f()  
# Start f()  
# Start g()  
# End g()  
# End f()

🡪

由于相同的变量名可以在不同的命名空间中存在。当我们在Python中访问某一个变量名的时候，Python便需要在scope中找寻具体该变量名指代哪个命名空间中的变量。

规则：

LEGB rules



假设该变量在某个方法中被访问，则先从local namespace中寻找。然后在包裹此方法的父方法enclosing space中寻找。如果还没有则到全局命名空间中寻找，如果还没有则到build-in命名空间中寻找。如果在build-in命名空间中都无法找到，则抛出NameError Exception. 采取这种inside-out的方式找寻。

🡪

**如果要在函数的局部命名空间中，改变(获取)处于全局变量的同名变量。则原则和将参数传入函数中，其形参值得改变是否会改变实参值的原则一样。**

如果是非可变类型，例如int, string等。则表现为PassByValue:

例如：

>>> x = 20 # x为int,非可变类型  
>>> def f():  
... x = 40 # 由于为非可变类型。所以x=40将创建一个x处于local space中。不会改变全局命名空间下的x  
... print(x)  
...  
  
>>> f()  
40  
>>> x  
20

如果是可变类型，例如数组，字典，元组等。则如果是部分赋值则表现为PassByReference,如果是整体赋值，则表现为PassByValue

例如：

>>> my\_list = ['foo', 'bar', 'baz'] # my\_list为可变对象，列表  
>>> def f():  
... my\_list[1] = 'quux' # 由于是部分赋值，所以可以获取到全局命名空间中的my\_list改变值。  
...  
>>> f()  
>>> my\_list  
['foo', 'quux', 'baz']

>>> my\_list = ['foo', 'bar', 'baz'] # my\_list为可变对象，列表  
>>> def f():  
... my\_list = ['qux', 'quux'] # 由于是整体赋值，所以Python会在局部命名空间中创建my\_list.而不是获取到全局命名空间中的my\_list  
...  
>>> f()  
>>> my\_list  
['foo', 'bar', 'baz']

🡪

Global关键字。Global关键字会强制指明变量指向全局命名空间中的变量。

例如：

>>> x = 20  
>>> def f():  
... global x  
... x = 40 # 由于是非可变类型，默认情况下会在本地命名空间中新建变量x.由于使用了global x，所以x将指向全局命名空间中的变量x.  
... print(x)  
...  
  
>>> f()  
40  
>>> x  
40

🡪

Nonlocal关键字。Nonlocal关键字会强制指明变量指向**最近的上一层**封闭式命名空间中的变量。

例如：

>>> def f():  
... x = 20  
...  
... def g():  
... nonlocal x # 默认情况下，由于x是非可变变量。所以会在本地g()的命名空间中创建变量x.并且使用global x也不行，global x会将变量x强制指到全局变量中。而我们想要的是指向封闭命名空间的x.所以使用nonlocal,它将指向最近的上一层封闭式命名空间的变量。  
... x = 40  
...  
... g()  
... print(x)  
...  
  
>>> f()  
40