在Python程序中使用变量时，Python创建、修改或查找变量名都是在命名空间（一个保存变量名的地方）中进行的。

Python将一个变量被赋值的地点关联为一个特定的命名空间。变量赋值的地方决定了这个变量存在于哪个命名空间，也是它可见的范围。

Python中，模块（Module），类（Class），函数（def）会产生新的作用域，if-else、for、while、try-except-finally等语句块中不会产生作用域，它们中定义的变量具有全局作用域。

例：程序variable\_scope\_test/variable\_scope\_test1.py

languages = ['English', 'Chinese']  
for language in languages:  
 study = language  
  
print(study) # OK  
print(language) # OK  
  
if len(languages) >0:  
 money = '1000'  
else:  
 money = '0'  
print(money) # OK  
  
while len(languages) >0:  
 name = 'Tom'  
break  
print(name) # OK  
  
try:  
 result = 10 / 5  
except ZeroDivisionError:  
 result = 'Error'  
print(result) # OK

输出为：

Chinese

Chinese

1000

Tom

2.0

LEGB规则：

L：local，本地作用域。在Python中，def或lambda定义的函数产生本地作用域，每次对函数的调用都产生一个新的本地作用域，Python中允许递归调用，每次递归都会产生一个新的本地作用域。在函数中赋值的变量，除非通过global语句或nonlocal语句来声明，都是本地变量，只能在函数内部使用。

例：程序variable\_scope\_test/variable\_scope\_test2.py

x = 99  
  
  
def func(y):  
 z = x + y  
return z  
  
  
func(0)  
# print(z, y) # Error，z, y为func内部定义  
  
  
languages = ['English', 'Chinese']  
  
  
# 在函数内部以任何方式赋值一个变量，该变量都会  
# 被设定为本地变量，但在函数内部修改了一个对象  
# 则不会把变量划分为本地变量  
# func\_1()中的x是本地变量，与全局的x互不影响  
# languages是全局变量  
def func\_1():  
 x = 8 # 赋值变量  
languages.append('Russia') # 修改变量  
print("func\_1 x = ", x)  
print("func\_1 languages id ", id(languages))  
  
  
func\_1()  
print('x = ', x)  
print('languages id ', id(languages))

输出为：

func\_1 x = 8

func\_1 languages id 627752996104

x = 99

languages id 627752996104

E：函数嵌套时，上一层def或lambda的本地作用域。

例：程序variable\_scope\_test/variable\_scope\_test3.py

x = 99  
  
  
# f2()中的x就位于嵌套作用域内

# 但不允许在f2()修改x，只能访问  
def f1():  
 x = 88  
  
 def f2():  
 # x -= 66 # Error 不能修改

print('f2() x: ', x)  
 f2()  
  
  
f1()

输出为：

f2() x: 88

在函数中，返回一个嵌套的函数，其能够记住嵌套作用域的变量的值。

例：程序variable\_scope\_test/factory.py

def maker(n):  
def action(p):  
return p \*\* n  
return action  
  
  
f = maker(2)  
g = maker(3)  
print(f)  
print(f(3)) # 9 3 \*\* 2  
print(f(4)) # 16 4 \*\* 2  
print()  
print(g)  
print(g(3)) # 27 3 \*\* 3

f记住的n为2，所以其计算平方，g记住的n为3，其计算立方。

G：全局作用域，一个.py文件中的顶层的变量对于这个文件内部的代码而言是全局的。如果希望在其他文件中使用，需要以导入模块的方式。全局变量是位于模块文件的顶层的变量。

例：程序udacity\_test/udacity\_test3.py

egg\_count = 0  
t1 = (1, 2)  
t1 += (3,)  
print(t1)  
l1 = [1, 2]  
l1 += [3]  
print(l1)  
  
  
# 在函数中不能对全局变量进行复合运算

# 使用global语句后，就可以进行复合运算  
def buy\_eggs():  
print()  
# l1 += [3] # error  
 # t1 += (1,) # error  
 # egg\_count += 12 # error  
  
  
buy\_eggs()

B：内置作用域，python的内置模块，需要import builtins

Python中搜索变量的顺序：L->E->G->B。

global语句：

例：variable\_scope\_test/variable\_scope\_test4.py

x = 88  
  
  
# 外部之前未定义，直接  
# 在函数内部使用global z  
# 调用函数后，z也可以在外部使用  
def func():  
global x  
 x = 99  
global z  
 z = 100  
  
  
func()  
print('x = ', x)  
print('z = ', z)

输出为：

x = 99

z = 100

如果不调用func()，z不可用。

nonlocal语句：函数内部赋值在嵌套作用域的变量，与global的功能类似，不同的是，nonlocal声明的变量，必须之前已经定义在嵌套作用域中。

例：variable\_scope\_test/variable\_scope\_test5.py

def tester(start):  
 state = start  
  
 def nested(label):  
 nonlocal state  
 print(label, state)  
 state += 1  
 return nested  
  
  
f = tester(0)  
f('spam') # spam 0  
f('ham') # ham 1  
f('egg') # egg 2  
print()  
  
g = tester(42)  
g('spam') # spam 42  
g('ham') # ham 43  
g('egg') # egg 44

从输出可以看出nested()函数中修改了嵌套作用域的变量。