Python 变量和作用域

在程序中使用变量名时， Python 创建 改变 或查找变量名都是在命名空间中进行的， 也可以把命名空间称为作用域， 代码中变量名被赋值的位置决定了这个变量能被访问到的范围

注意
1.一个 def 内定义的变量名只能被 def 内的代码使用 不能在函数外部引用这样的变量名 （前提是没将变量定义为 global）

2.def 之内的变量名 跟 def 之外的变量名并不冲突 也就是说如果一个变量 x 一个定义在 def 之外一个定义在 def 之内 两个 x 是不同的变量

1)如果一个变量在 def 之内赋值 则它被定义在函数之内
2)如果一个变量在一个嵌套的 def 中赋值 对于嵌套函数来说 它是非本地的
3)如果在 def 之外赋值 它就是全局变量

变量的作用域完全是由变量在程序源代码中的位置决定的

例如

```
x = 99
def f1():
    x = 88
    print x

f1()
print x
```

上述例子中两个变量名都是 x 但是作用域可以把他们区分开 实际上函数的作用域有助于防止程序中变量名的冲突 并有助于帮助函数成为更加独立的程序单元

作用域法则：

1. 每一个模块(.py 文件)都是一个全局作用域 ，其他 py 文件中的全局变量，对于本 py 文件来说 相当于是一个模块中的属性 如
   demo1.py demo2.py

   vim demo1.py
   from demo2 import *
   此时 demo2 中的全局变量 对与 demo1 来说就是 demo2 模块里面的属性

2. 全局作用域的范围仅限于单个文件
3. 所有的变量名都可以归纳为 本地 全局 或 内置的
4. 函数中赋值的变量除非用 global 或者是 nolocal 声明 否则都为本地变量

变量名解析规则：LE(嵌套函数中)GB

在函数中使用未认证的变量名时 Python 搜索 4 个作用域 L E G B
L: local #本地作用域
E：Enclosing function #上一层结构中的 def 或 lambda 应用在嵌套函数中
G: global #全局
B：built-in(Python) #内置作用域

在使用变量时 python 对变量的查找是在第一处能够找到这个变量名的地方停下来 如果找不到则报 变量不存在错误

Global 语句
它是 python 中看起来有些像声明类型的语句 但是它并不是一个类型声明或者是大小声明 而是一个命名空间的声明 它告诉 python 打算生成一个或多个全局变量名,也就是一个存在于整个模块内部作用域的变量名

全局变量名总结:
1. 函数外定义的全局变量在函数内可以直接使用
2. 全局变量如果是在函数内被定义必须要经过声明
3. 全局变量是位于模块内(.py 文件)内部的顶层变量名

例:

```
x = 99
def f2():
    global x  ##global 声明可以在 def 之内可以引用 def 之外的 x
    x = 88
f2()
print x
```

作用域与嵌套函数

第一个嵌套函数

```
def f1():
    x = 77
    def f2():
        print x
    f2()
f1()  ##打印出 77
```

因为在函数 f2 中没有对应的 x 的赋值，也就是 local 中没找到 x，需要

向上一层函数也就是 f1 中去找变量 x ， 可以找到则可以返回它的值

闭包(工厂函数)

```
def f1(x):
    def f2(y):
        return x * y
    return f2


r = f1(2)
print r
print r(3)
```

输出
<function f2 at 0x10d901f50>
6

定义了一个外部函数 f1， 这个 f1 返回了嵌套的函数 f2 但是并没有调用 f2,

r(3)
r(4) ##这将会调用内嵌函数 内嵌函数记住了整数 2 也就是调用 f1 时候传入的参数 2， 并且将它与调用 f2 时候传入的参数 3 相乘 得到最终的结果 6

Nolocal (python3 中新特性 只在一个函数中有意义)

例:

```
    def f1(start):
        state = start
        def f2(label):
            return label, state
        return f2


x = f1('apple')
print(x(2))

结果:
#(2, apple)
```

在 python2 中默认不允许修改嵌套 def 中的变量

例:

```
def f1(x):
    state = x
    def f2(y):
        return state, y
        state += 1
    return f2

x = f1(3)
print x(2)
```

```
[root@slave3 ~]# python demo1.py
Traceback (most recent call last):
  File "demo1.py", line 9, in <module>
    print x(2)
  File "demo1.py", line 4, in f2
    return state, y
UnboundLocalError: local variable 'state' referenced before assignment  ##发生报错
```

在 python3 中实现上面代码

```
    def f1(start):
    state = start
    def f2(label):
        nonlocal state
        state += 1
        return  label, state

    return f2


x = f1(2)
print(x('apple'))
print(x('banana'))


#('apple', 3)
#('banana', 4)
```

作用域的实例练习

下面的代码会输出什么？为什么？

```
x  = 'spam'
def f1():
    print x

f1()

def f2():
    x =  'hello'
    return x
print f2()
print x
```

```
x =  ‘a’
def f4():
    global x
    x = ’f4 word’
print f4()
print x


x = ’spam’
def f5():
    x = ’f5’
    def nested():
        print x
    nested()
f5()
print x
```