Python变量和作用域

一.作用域

在程序中使用变量名时, Python创建 改变 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.

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.

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

In [1]:
```
#示例代码1:

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

f1()
print x
```

```
88
99
```

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.

二.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 *

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.

三.变量名解析规则

变量名解析规则：LE(嵌套函数中)GB

在函数中使用变量名时, 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 #全局作用域

B：built-in(Python) #Python的内置作用域

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global声明

3) 全局变量是位于模块内(.py文件)内部的顶层变量名

In [3]:
```
#示例代码2:

x = 99
def f2():
    global x
    x = 88
f2()
print x
```

```
88
```

In [4]:
```
#示例代码3: 作用域与嵌套函数

def f1():
    x = 77
    def f2():
        print x
    f2()
f1()
```

```
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.

In [8]:
```
#示例代码4: 函数的闭包
def f1(x):
    def f2(y):
        return x * y
    return f2

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

```
<function f2 at 0x7f94c3223de8>
6
```

说明:

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 = f1(2)

print r(3) 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

In [1]:
```
#示例代码5: nonlocal, 判断内层函数被调用的次数(Python3支持,  Python2报错)

def outer(x):
    c = 0
    def inner(y):
        nonlocal c
        c = c + 1
        print(str(c) + '  test')
        return x + y
    return inner

x = outer(3)

print(x(1))
print(x(2))
print(x(10))
print(x(11))
```

```
1  test
4
2  test
5
3  test
13
4  test
14
```

In [4]:
```
# 在函数内部试图修改全局变量所遇到的问题

x = 1
def f1():
    x += 1
    print(x)

f1()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-4-5c1e954619b1> in <module>
      6     print(x)
      7
----> 8 f1()

<ipython-input-4-5c1e954619b1> in f1()
      3 x = 1
      4 def f1():
----> 5     x += 1
      6     print(x)
      7

UnboundLocalError: local variable 'x' referenced before assignment
```

导致上述问题的原因是, 函数内部对于全局变量只有读取权限没有修改权限, 如果想在函数中修改全局变量的值, 应该在函数内部对变量用global修饰, 例子如下:

In [5]:
```
x = 1
def f1():
    global x
    x = x + 1
    print(x)
f1()
```

```
2
```

In [15]:
```
#示例代码6: 变量作用域相关练习

x  = 'spam'
def f1():
       print x
f1()
print '#' * 10

def f2():
    x =  'hello'
    return x
print f2()
print x
print '#' * 10

x = 'a'
def f4():
    global x
    x = 'f4 word'
print f4()
print x
print '#' * 10

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

```
spam
##########
hello
spam
##########
None
f4 word
##########
f5
spam
```

In [ ]: