nonlocal

在使用函数闭包时，只有 可变对象 才能在运行中改变值，如：

```
def f():
    x = []
    def g():
        x.append(1)
    g()
    return x

print(f())
```

执行结果为：

```
[1]
```

而如果使用 不可变对象 ， 并尝试对其赋值：

```
def f():
    x = 0
    def g():
        x = 1
    g()
    return x

print(f())
```

执行结果为：

```
0
```

这是因为 int 是个不可变对象，当尝试对其赋值时， Python 会在一个新的内存地址写入值。

或者并对其引用，并尝试对齐赋值：

```
def f():
    x = 0
    def g():
        if x == 0:
            x = 1
    g()
    return x

print(f())
```

执行结果为：

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

在这个例子中， `g()` 中 x 会被直接认为时局部变量，而不形成一个闭包。

但是如果只读取值，不写入值，是没有问题的：

```
def f():
    x = 0
    def g():
        if x == 0:
            print('helloworld')
    g()
    return x

print(f())
```

执行结果为：

```
helloworld
0
```

如果对不可变对象的闭包中既需要读取值，又需要写入值，可以使用 `nonlocal` 关键字：

```
def f():
    x = 0
    def g():
        nonlocal x
        if x == 0:
            x = 1
    g()
    return x

print(f())
```

执行结果为：

```
1
```