# 闭包

* 绑定了外部作用域的变量的函数
* 即使程序离开外部作用域，如果闭包仍然可见，绑定变量不会销毁
* 每次运行外部函数都会重新创建闭包
* 闭包：引用了外部自由变量的函数
* 自由变量：不在当前函数定义的变量
* 特性：自由变量会和闭包函数同时存在

from functools import wraps

def cache(func):
 store = {}
 @wraps(func)
 def \_(n):
 if n in store:
 return store[n]
 else:
 res = func(n)
 store[n] = res
 return res
 return \_

@cache
def f(n):
 if n <= 1:
 return 1
 return f(n-1) + f(n-2)

为什么用@wraps(func)

因为当使用装饰器装饰一个函数时，函数本身就已经是一个新的函数；即函数名称或属性产生了变化。所以在python的functools模块中提供了wraps装饰函数来确保原函数在使用装饰器时不改变自身的函数名及应有属性。 所以在装饰器的编写中建议加入wraps确保被装饰的函数不会因装饰器带来异常情况。 https://blog.csdn.net/yuyexiaohan/article/details/82860807

## 习题

### 解释下什么是闭包

在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包。

简单的说，如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包(closure)。来看一个简单的例子:

>>>def addx(x):
>>> def adder(y): return x + y
>>> return adder
>>> c = addx(8)
>>> type(c)
<type 'function'>
>>> c.\_\_name\_\_
'adder'
>>> c(10)
18