day11：lambda表达式，闭包，函数式编程

笔记本：	Python基础		
创建时间：	2018/6/21 9:33	**更新时间：**	2018/7/2 19:38
作者：	liuchang_0412@163.com		
URL：	https://blog.csdn.net/Yeoman92/article/details/67636060		

1.lambda表达式（又称匿名函数对象）：创建一个匿名函数对象，同def类似，但不提供函数名

①语法：**lambda 参数1,参数2,...：表达式**（[ ]内的部分可以省略）

②示例：

```
def myadd(x,y):
    return x+y
#可以改写为
myadd = lambda x,y : x+y

#练习看懂如下代码：
def operator(fn,x,y):
    return fn(x,y)

operator((lambda a,b:a+b), 100, 200)    #返回值是多少？ --> 300
operator((lambda a,b:a*b), 100, 200)    #返回值是多少？ --> 20000
```

③语法说明：

1.lambda只是一个表达式，用来创建一个函数对象

2.当lambda表达式调用时，返回的是冒号后表达式的值

3.lambda表达式创建的函数只能包含一条语句

4.lambda比函数简单且可以随时创建和销毁，有利于减少程序的耦合度

```
#练习：写一个lambda表达式，求两个变量的最大值
mymax2 = lambda x, y : x if x > y else y
```

2.globals( ) 和 locals( )函数

①globals( )返回当前全局作用域内变量的字典

②locals( )返回当前局部作用域内变量的字典

示例：见code2.py

3.eval( )和exec( )函数：

①eval( )：把一个字符串当成一个表达式来执行，返回表达式执行后的结果

格式：eval(source, globals=None, locals=None)

示例：

```
x = 100
y = 200
a = eval("x+y")
print(a)    #300
```

②exec( )：把一个字符串成一个程序来执行

格式：exec(source, globals=None, locals=None)（globals与locals作用同eval( ))

示例：

```
x = 100
y = 200
s = "print('hello:',x,y)"
exec(s)    #hello: 100 200

gs = {"x":10,"y":20}
ls = {"x":1,"y":2}
exec("z=x+y", gs , ls)
print(ls)    #{"x":1,"y":2,"z":3}
```

#练习：自己写一个程序的解释进行器，用于解释我们自己输入的程序

```
$ ./myprog
请输入程序：>>>> x = 100<回车>
请输入程序：>>>> y = 200<回车>
请输入程序：>>>> print("x+y=",x+y)<回车>
x+y= 300
请输入程序：>>>>
```

4.闭包 closure：将组成函数的语句和这些语句的执行环境打包在一起时，得到的对象称为闭包

说明：如果一个内嵌函数访问函数外部作用域的变量，则这个函数就是闭包

闭包的作用：

1.当闭包执行完后，仍然能够保持住当前的运行环境

2.闭包可以根据外部作用域的局部变量来得到不同的结果

示例：

```
def make_power(x):
    def fn(arg):
        return arg ** x
    return fn

f2 = make_power(2)    #f2所绑定的函数称为闭包    #y = f2(100) --> 100**2
```

便于理解闭包的网上示例：

https://blog.csdn.net/Yeoman92/article/details/67636060

5.函数式编程：是指用一系列函数解决问题

①函数是编程的概念：

1.每一个函数完成细小的功能，一系列函数的任意组合可以完成大问题

2.函数仅接收收入并产生输出，不包含任何可能影响输出的内部状态（不引用外部变量）

②函数式编程的优点：

1.易于测试

2.易于调试

3.更高的生产率

4.模块化

5.逻辑可证

③函数的可重入性：可重入性是指输入一定，则输出必须一定

例：#不可重入的函数：

```
y=200
def myadd(x):
    return x+y
print(myadd(10))    #210
y=300
print(myadd(10))    #310
    #可重入的函数
def add2(x,y):
    return x+y
```

⑤高阶函数：满足下列条件中的一个的函数即为高阶函数

a.函数接受一个或多个函数作为参数传入

b.函数返回一个函数

6.Python中内置(builtins)的高阶函数：

①map( )函数：

格式：map(func, *iterable)　用函数和可迭代对象中的每个元素作为参数，计算出新的可迭代对象，当最短的一个可迭代对象完成迭代后，此迭代生成结束,返回map object。

示例：

```
def power2(x):
    return x ** 2
#生成一个迭代器，此迭代器可以生成1~9的自然数平方(1 4 9 16 ... )
mit = map(power2, range(1,10))
for x in mit:
    print(x,end= ' ')
```

#练习：求1**9+2**8+3**7+...+9**1的和，用函数式编程做。

②filter( )函数：筛选序列中的数据，返回一个可迭代对象，此可迭代对象将对iterable进行筛选

格式：filter(function or None, iterable)

说明：function 将对iterable中的每个元素进行求值，返回False则将此数据丢弃，返回True，则保留此数据

示例：

```
L=[x for x in range(10)]    #[0,1,2,3,..,9]
def isodd(x):    #如果为奇数返回True
    return x%2 == 1

L2=list(filter(isodd,range(10)))
```

③sorted( )函数：将原可迭代对象的数据进行排序，生成排序后的列表

格式：sorted(iterable, key=None, reverse=False)

说明：key函数用来提供一个值，这个值将作为排序的依据

示例：

```
L=[5,-2,-4,0,3,1]
L2=sorted(L)                    #[-4,-2,0,1,3,5]
L2=sorted(L,reverse=True)       #[5,3,1,...]
L2=sorted(L,key=abs)            #[0,1,-2,3,-4,5]
```

7.递归函数 recursion：函数直接或间接的调用自身

说明：递归一定要控制递归的层数，当符合某一条件时要终止递归调用，几乎所有递归都能用while循环来替代

优点：

可以把问题简单化，让思路更清晰，代码更简洁

缺点：

递归因系统环境影响大，当递归深度太大时，可能会得到不可预知的结果

示例：

```
def f( ) :
    print("hello")
    f( )
f( )        #调用函数
print("递归完成")
```

#练习：

1.用filter函数将1~100之间的所有素数prime放入到列表中并打印

2.用递归方式计算1+2+3+...+n的和（已完成）

3.用函数式编程算出1!+2!+3!+...+n!。

4.改写之前学生信息的程序，每个人的信息有：姓名:name,年龄:age,成绩:score

输入5个学生信息，做如下操作：

a.按成绩从高到低打印学生信息

b.按年龄从高至低打印学生信息

c.按年龄从低至高打印学生信息

d.按原来输入顺序打印学生信息