函数与过程

函数的声明--函数的实现--函数的返回值----不准确
1. 函数
        函数声明+加函数的返回值
2. 过程
        函数的实现

- 函数的返回值

关键字：return
    return 值
1. 任何函数都有返回值
2. 如果不加return ，系统会默认加一个return None
3. 如果书写了return，但是没有给返回值，系统会加一个None
4. return后可以跟任何数据
5. return后可以跟一个表达式---最终结果
6. return后面可以跟一个函数的调用
7. return同级下面代码不会执行，代表着函数的结束

- 函数变量的作用域

作用域--作用范围--命名空间
1. 局部变量
        1. 定义在函数内部的变量
        2. 先赋值在使用
        3. 从定义开始到包含他的代码块结束
        4. 在外部无法访问
        5. 如果全局变量和局部变量发生了命名冲突，以局部变量优先
2. 全局变量
        1. 定义在文件的顶头（没有缩格）
        2. 整个文件
        3. 先赋值在使用（如果以脚本形式运行，运行的代码可能会截断作用域）

- global关键字

使用方法：
    global 全局变量名
1. global声明的是全局变量
        从本行开始该函数内部的变量都是全局变量

```
a=100
print(a)
def fun():
    global a  #
    a=200
fun()
```

```
            print(a)
2. nonlocal声明的是局部变量
    关键字：nonlocal
        nonlocal 局部变量
        从本行开始该函数的变量用的变量都是外层的局部变量（向外一层）
        def fun1():
            a=200
            def fun2():
                nonlocal a
                a=300
                print(a,'1')
                def fun3():
                    nonlocal a
                    a=400
                    print(a,'3')
                fun3()
                print(a,'4')

            fun2()
            print(a,'2')
        fun1()
```

- 补充

```
在函数内部不能直接进行a=a+1--a+=1
a=100
def fun():
    a=a+1
    print(a)

fun()
解决办法
    1. 在函数内部定义一个局部变量
    2. global a（前提是必须要有全局变量）
-----------------------------------------
def fun1():
    a=100
    def fun2():
        a=a+1
        print(a)
    fun2()
fun1()
解决方案：
    1.nonlocal 局部变量
    2. 在内层函数中定义一个局部变量
    3. 通过列表解决（python2版本常用）
        def fun1():
            a=[100]
            def fun2():

                a[0]=a[0]+1
                print(a)
```

```
            fun2()
            print(a[0])
        fun1()
```

- 内嵌函数

```
在函数内部定义的函数
def fun1():
    print('this is fun1')
    def fun2():
        print('this is fun2')

    fun2()
fun1()
```

- 闭包

```
闭包：是函数式编程重要语法结构---编程范式
编程范式：对代码进行提炼和抽象概括---重用性高
```

- 闭包的条件

```
1. 内嵌函数
2. 内层函数使用外层函数的局部变量
3. 外层函数返回内层函数的函数对象（函数名）
以下不是闭包：
def fun1():
    a=10
    def fun2(b):
        c=a+b
        print(c)
    return fun2(10)
fun1()
--------------
def fun1():
    a=10
    def fun2(b):
        c=b*2
        print(c)
    return fun2
fun1()(10)
```

- 闭包的 作用

1. 可以定义更少的参数--传递更多的参数---更加安全

```
def fun1():
a=10
def fun2(b):

    c=b*2
    print(c)

return fun2
fun1()(10)
```

2. 间接的访问内部函数
3. 用于集群操作

- 闭包的优点

1. 避免使用全局变量---函数污染

```
    a=[[1,2,3],[4,5,6]]

def fun():
    a[0][0]=99
    print(a)
fun()

def fun2():
    print(a)

fun2()
```

2. 可以提供部分数据的隐藏
3. 可以提供更加优雅的面相对象实现

```
# 计算某一个固定值和其他数的相加结果---

def fun1(a):
    def fun2(s):
        return s+a
    return fun2

c=fun1(10)
print(c(1))
print(c(2))
```

- Lambda表达式

1. 匿名函数
    没有名字的函数
2. 使用时机：
    只是用一次的时候
3. lambda语法：
    关键字：lambda
    lambda 参数1，参数2：返回值

4. 更加简洁，优雅
    print((lambda x:x)(10))
5. 优先级最低，运算效率低

- lambda的作用

1. 只需要使用一次
2. 简化代码提高可读性

- 函数嵌套调用

在函数内部调用函数

```
def fun1():
    print('hehe')  #所有书籍

def fun2():
    print('hello~~~')   #添加书籍

    fun1()

fun2()
```