

# Table of Contents

Python基础	0
函数	1
函数介绍	1.1
函数定义、调用	1.2
函数的文档说明	1.3
函数参数(一)	1.4
函数返回值(一)	1.5
4种函数的类型	1.6
函数的嵌套调用	1.7
函数应用：打印图形和数学计算	1.8
局部变量	1.9
全局变量	1.10
函数应用：学生管理系统	1.11
函数返回值(二)	1.12
函数参数(二)	1.13
递归函数	1.14
匿名函数	1.15
函数使用注意事项	1.16
作业	1.17





# 函数介绍

## <1>什么是函数

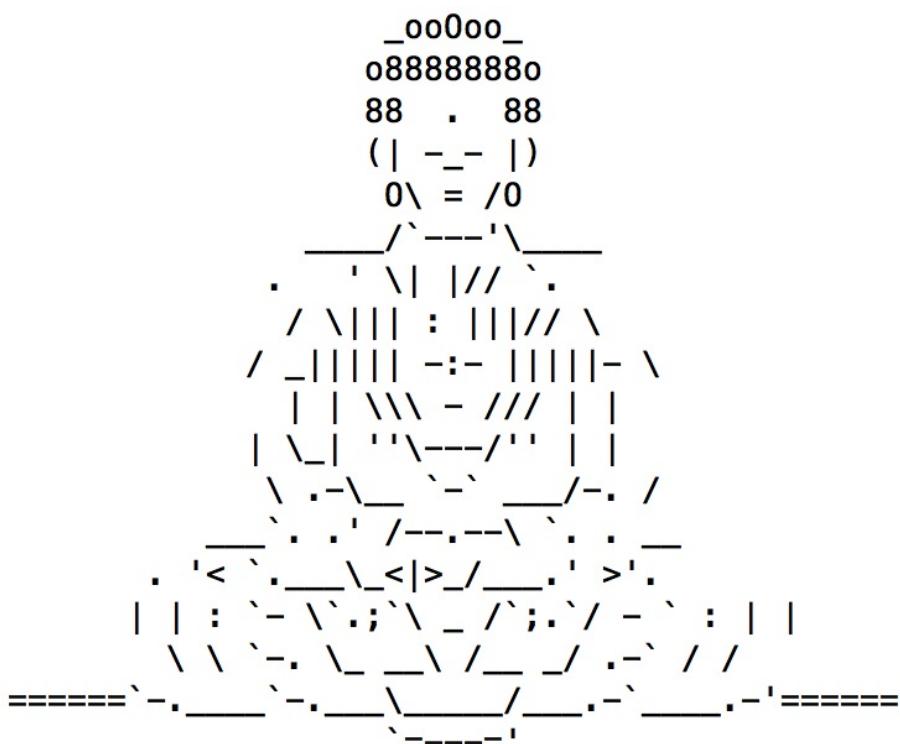
请看如下代码：

```

print "          _oo0oo_ "
print "          o8888888o "
print "          88 . 88 "
print "          (| -_- |) "
print "          0\\ = /0 "
print "          ____/_---'\\ \____ "
print "          . . ' \| | | / / ` . "
print "          / \| ||| : ||| / / \| "
print "          / _||| | -:- ||| | - \| "
print "          | | \| \| \| - / / | | "
print "          | \|_| ' \| \| ---/ ' | | "
print "          \| . - \|_ \| -` _ \| / . / "
print "          _ \| . . ' / - - - \| ` . . _ "
print "          . "" ' < ` . __ \| < | > / __ . ' > ' "" . "
print "          | | : ^ - \| \ . ; \| _ / ^ ; . ; ^ - ^ : | | "
print "          \| \| \| ^ - . \| _ \| \| / _ \| . - ^ / / "
print "          =====^ - . __ ^ - . __ \| \| / _ \| . - ^ _ \| . - ' ===== "
print "          "
print "          ..... "
print "          佛祖镇楼           BUG辟易 "
print "          佛曰： "
print "          写字楼里写字间，写字间里程序员； "
print "          程序人员写程序，又拿程序换酒钱。 "
print "          酒醒只在网上坐，酒醉还来网下眠； "
print "          酒醉酒醒日复日，网上网下年复年。 "
print "          但愿老死电脑间，不愿鞠躬老板前； "
print "          奔驰宝马贵者趣，公交自行程序员。 "
print "          别人笑我忒疯癫，我笑自己命太贱； "
print "          不见满街漂亮妹，哪个归得程序员？ "

```

运行后的现象：



.....  
佛祖镇楼

BUG辟易

佛曰：

写字楼里写字间，写字间里程序员；  
程序人员写程序，又拿程序换酒钱。  
酒醒只在网上坐，酒醉还来网下眠；  
酒醉酒醒日复日，网上网下年复年。  
但愿老死电脑间，不愿鞠躬老板前；  
奔驰宝马贵者趣，公交自行程序员。  
别人笑我忒疯癫，我笑自己命太贱；  
不见满街漂亮妹，哪个归得程序员？

想一想：

如果一个程序在不同的地方需要输出“佛祖镇楼”，程序应该怎样设计？

```
if 条件1:  
    输出‘佛祖镇楼’
```

... (省略) ...

```
if 条件2:  
    输出‘佛祖镇楼’
```

... (省略) ...

如果需要输出多次，是否意味着要编写这块代码多次呢？

## 小总结：

- 如果在开发程序时，需要某块代码多次，但是为了提高编写的效率以及代码的重用，所以把具有独立功能的代码块组织为一个小模块，这就是函数

# 函数定义和调用

## <1> 定义函数

定义函数的格式如下：

```
def 函数名():
    代码
```

demo:

```
# 定义一个函数，能够完成打印信息的功能
def printInfo():
    print '-----'
    print '        人生苦短，我用Python'
    print '-----'
```

## <2> 调用函数

定义了函数之后，就相当于有了一个具有某些功能的代码，想要让这些代码能够执行，需要调用它

调用函数很简单的，通过 **函数名()** 即可完成调用

demo:

```
# 定义完函数后，函数是不会自动执行的，需要调用它才可以
printInfo()
```

## <3>练一练

要求：定义一个函数，能够输出自己的姓名和年龄，并且调用这个函数让它执行

- 使用**def**定义函数
- 编写完函数之后，通过**函数名()**进行调用

# 函数的文档说明

```
>>> def test(a,b):  
...     "用来完成对2个数求和"  
...     print("%d"%(a+b))  
...  
>>>  
>>> test(11,22)  
33
```

如果执行，以下代码

```
>>> help(test)
```

能够看到test函数的相关说明

```
Help on function test in module __main__:  
  
test(a, b)  
    用来完成对2个数求和  
(END)
```

# 函数参数(一)

思考一个问题，如下：

现在需要定义一个函数，这个函数能够完成2个数的加法运算，并且把结果打印出来，该怎样设计？下面的代码可以吗？有什么缺陷吗？

```
def add2num():
    a = 11
    b = 22
    c = a+b
    print c
```

为了让一个函数更通用，即想让它计算哪两个数的和，就让它计算哪两个数的和，在定义函数的时候可以让函数接收数据，就解决了这个问题，这就是函数的参数

## <1> 定义带有参数的函数

示例如下：

```
def add2num(a, b):
    c = a+b
    print c
```

## <2> 调用带有参数的函数

以调用上面的add2num(a, b)函数为例：

```
def add2num(a, b):  
    c = a+b  
    print c
```

add2num(11, 22) #调用带有参数的函数时，需要在小括号中，传递数据

调用带有参数函数的运行过程：

→ #定义接收2个参数的函数  
def add2num(a, b):  
 c = a+b  
 print c

#调用带有参数的函数  
add2num(110, 22)

终端

## <3> 练一练

要求：定义一个函数，完成前2个数完成加法运算，然后对第3个数，进行减法；然后调用这个函数

- 使用def定义函数，要注意有3个参数
- 调用的时候，这个函数定义时有几个参数，那么就需要传递几个参数

## <4> 调用函数时参数的顺序

```
>>> def test(a,b):
...     print(a,b)
...
>>> test(1,2)
1 2
>>> test(b=1,a=2)
2 1
>>>
>>> test(b=1,2)
File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
>>>
>>>
```

## <4> 小总结

- 定义时小括号中的参数，用来接收参数用的，称为“形参”
- 调用时小括号中的参数，用来传递给函数用的，称为“实参”

# 函数返回值(一)

## <1>“返回值”介绍

现实生活中的场景:

我给儿子10块钱，让他给我买包烟。这个例子中，10块钱是我给儿子的，就相当于调用函数时传递到参数，让儿子买烟这个事情最终的目标是，让他把烟给你带回来然后给你对么，，，此时烟就是返回值

开发中的场景:

定义了一个函数，完成了获取室内温度，想一想是不是应该把这个结果给调用者，只有调用者拥有了这个返回值，才能够根据当前的温度做适当的调整

综上所述:

- 所谓“返回值”，就是程序中函数完成一件事情后，最后给调用者的结果

## <2>带有返回值的函数

想要在函数中把结果返回给调用者，需要在函数中使用return

如下示例:

```
def add2num(a, b):  
    c = a+b  
    return c
```

或者

```
def add2num(a, b):  
    return a+b
```

## <3>保存函数的返回值

在本小节刚开始的时候，说过的“买烟”的例子中，最后儿子给你烟时，你一定是从儿子手中接过来 对么，程序也是如此，如果一个函数返回了一个数据，那么想要用这个数据，那么就需要保存

保存函数的返回值示例如下：

```
#定义函数
def add2num(a, b):
    return a+b

#调用函数，顺便保存函数的返回值
result = add2num(100,98)

#因为result已经保存了add2num的返回值，所以接下来就可以使用了
print result
```

结果：

```
198
```

# 4种函数的类型

函数根据有没有参数，有没有返回值，可以相互组合，一共有4种

- 无参数，无返回值
- 无参数，有返回值
- 有参数，无返回值
- 有参数，有返回值

## <1>无参数，无返回值的函数

此类函数，不能接收参数，也没有返回值，一般情况下，打印提示灯类似的功能，使用这类的函数

```
def printMenu():
    print('-----')
    print('      xx涮涮锅 点菜系统')
    print('')
    print('  1. 羊肉涮涮锅')
    print('  2. 牛肉涮涮锅')
    print('  3. 猪肉涮涮锅')
    print('-----')
```

结果：

-----  
xx涮涮锅 点菜系统  
1. 羊肉涮涮锅  
2. 牛肉涮涮锅  
3. 猪肉涮涮锅  
-----

## <2>无参数，有返回值的函数

此类函数，不能接收参数，但是可以返回某个数据，一般情况下，像采集数据，用此类函数

```
# 获取温度
def getTemperature():
    #这里是获取温度的一些处理过程
    #为了简单起见，先模拟返回一个数据
    return 24

temperature = getTemperature()
print('当前的温度为：%d'%temperature)
```

结果：

```
当前的温度为： 24
```

## <3>有参数，无返回值的函数

此类函数，能接收参数，但不可以返回数据，一般情况下，对某些变量设置数据而不需结果时，用此类函数

## <4>有参数，有返回值的函数

此类函数，不仅能接收参数，还可以返回某个数据，一般情况下，像数据处理并需要结果的应用，用此类函数

```
# 计算1~num的累积和
def calculateNum(num):

    result = 0
    i = 1
    while i<=num:

        result = result + i

        i+=1

    return result

result = calculateNum(100)
print('1~100的累积和为:%d'%result)
```

结果:

```
1~100的累积和为: 5050
```

## <5>小总结

- 函数根据有没有参数，有没有返回值可以相互组合
- 定义函数时，是根据实际的功能需求来设计的，所以不同开发人员编写的函数类型各不相同

# 函数的嵌套调用

```
def testB():
    print('---- testB start----')
    print('这里是testB函数执行的代码...(省略)...')
    print('---- testB end----')

def testA():

    print('---- testA start----')

    testB()

    print('---- testA end----')

testA()
```

结果：

```
---- testA start----
---- testB start----
这里是testB函数执行的代码...(省略)...
---- testB end----
---- testA end----
```

小总结：

- 一个函数里面又调用了另外一个函数，这就是所谓的函数嵌套调用

The diagram shows two Python functions, testA and testB, with their code and a red arrow indicating the call stack.

```
def testB():
    print('---- testB start----')
    print('这里是 testB函数执行的代码...(省略)...')
    print('---- testB end----')

def testA():
    print('---- testA start----')
    testB()
    print('---- testA end----')

testA()
```

A red arrow starts at the opening parenthesis of the first testB() call and points to the opening parenthesis of the second testB() call. Another red arrow starts at the opening parenthesis of the testA() call and points to the opening parenthesis of the first testB() call.

- 如果函数A中，调用了另外一个函数B，那么先把函数B中的任务都执行完毕之后才会回到上次 函数A执行的位置

# 函数应用：打印图形和数学计算

## 目标

- 感受函数的嵌套调用
- 感受程序设计的思路,复杂问题分解为简单问题

## 思考&实现1

1. 写一个函数打印一条横线
2. 打印自定义行数的横线

## 参考代码1

```
# 打印一条横线
def printOneLine():
    print("-"*30)

# 打印多条横线
def printNumLine(num):
    i=0

    # 因为printOneLine函数已经完成了打印横线的功能,
    # 只需要多次调用此函数即可
    while i<num:
        printOneLine()
        i+=1

printNumLine(3)
```

## 思考&实现2

1. 写一个函数求三个数的和
2. 写一个函数求三个数的平均值

## 参考代码2

```
# 求3个数的和
def sum3Number(a,b,c):
    return a+b+c # return 的后面可以是数值，也可是一个表达式

# 完成对3个数求平均值
def average3Number(a,b,c):

    # 因为sum3Number函数已经完成了3个数的就和，所以只需调用即可
    # 即把接收到的3个数，当做实参传递即可
    sumResult = sum3Number(a,b,c)
    aveResult = sumResult/3.0
    return aveResult

# 调用函数，完成对3个数求平均值
result = average3Number(11,2,55)
print("average is %d"%result)
```

# 局部变量

## <1>什么是局部变量

如下图所示：

The screenshot shows two windows. On the left is Sublime Text with a file named 'test.py'. The code defines two functions: test1() and test2(). In test1(), there is a local variable 'a' assigned the value 300. In test2(), there is another local variable 'a' assigned the value 400. Red arrows point from the variable declarations in both functions to the text '局部变量' (Local Variable) located between them. On the right is a terminal window showing the output of running the script: it prints three lines: '----test1--修改前--a=300', '----test1--修改后--a=200', and '----test3----a=400'.

```

test.py - Sublime Text
test.py
1 #coding=utf-8
2
3 def test1():
4     a = 300
5     print('----test1--修改前--a=%d'%a)
6     a = 200
7     print('----test1--修改后--a=%d'%a)
8
9 def test2():
10    a = 400
11    print('----test3----a=%d'%a)
12
13
14 # 调用函数
15 test1()
16 test2()
17

```

```

python@ubuntu:~/Desktop$ python3 test.py
----test1--修改前--a=300
----test1--修改后--a=200
----test3----a=400
python@ubuntu:~/Desktop$

```

## <2>小总结

- 局部变量，就是在函数内部定义的变量
- 不同的函数，可以定义相同的名字的局部变量，但是各用个的不会产生影响
- 局部变量的作用，为了临时保存数据需要在函数中定义变量来进行存储，这就是它的作用

# 全局变量

## <1>什么是全局变量

如果一个变量，既能在一个函数中使用，也能在其他的函数中使用，这样的变量就是 全局变量

demo如下：

```
# 定义全局变量
a = 100

def test1():
    print(a)

def test2():
    print(a)

# 调用函数
test1()
test2()
```

运行结果：

```
python@ubuntu:~/Desktop$ python3 test.py
100
100
```

## <2>全局变量和局部变量名字相同问题

看如下代码：

The screenshot shows a Sublime Text editor with a file named 'test.py'. The code defines a global variable 'a' and uses it within functions 'test1' and 'test2'. A red box highlights the assignment 'a = 100' with the text '全局变量' (Global Variable). Another red box highlights the assignment 'a = 300' with the text '局部变量' (Local Variable). To the right, a terminal window shows the execution of the script, demonstrating that changes made in one function do not affect the global variable.

```
test.py - Sublime Text
test.py
1 #coding=utf-8
2 # 定义全局变量
3 a = 100
4
5 def test1():
6     a = 300
7
8     print('----test1--修改前--a=%d'%a)
9     a = 200
10    print('----test1--修改后--a=%d'%a)
11
12 def test2():
13     print('----test3----a=%d'%a)
14
15
16
17 # 调用函数
18 test1()
19 test2()

python@ubuntu:~/Desktop$ python3 test.py
----test1--修改前--a=300
----test1--修改后--a=200
----test3----a=100
python@ubuntu:~/Desktop$
```

## <3>修改全局变量

既然全局变量，就是能够在所有的函数中进行使用，那么可否进行修改呢？

代码如下：

The screenshot shows a Sublime Text editor with the same 'test.py' code as before, but now includes a 'global a' statement in the 'test1' function. A red box highlights this statement with the text 'global a'. To the right, a terminal window shows the execution of the script, confirming that the modification of the global variable 'a' within 'test1' does not affect its value in 'test2'.

```
test.py - Sublime Text
test.py
1 #coding=utf-8
2 # 定义全局变量
3 a = 100
4
5 def test1():
6     global a
7
8     print('----test1--修改前--a=%d'%a)
9     a = 200
10    print('----test1--修改后--a=%d'%a)
11
12 def test2():
13     print('----test3----a=%d'%a)
14
15
16
17 # 调用函数
18 test1()
19 test2()

python@ubuntu:~/Desktop$ python3 test.py
----test1--修改前--a=100
----test1--修改后--a=200
----test3----a=200
python@ubuntu:~/Desktop$
```

程序ok,  
没有出错

## <4>总结1：

- 在函数外边定义的变量叫做 全局变量
- 全局变量能够在所有的函数中进行访问
- 如果在函数中修改全局变量，那么就需要使用 `global` 进行声明，否则

出错

- 如果全局变量的名字和局部变量的名字相同，那么使用的是局部变量的，小技巧 强龙不压地头蛇

## <5>可变类型的全局变量

```

>>> a = 1
>>> def f():
...     a += 1
...     print a
...
>>> f()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
UnboundLocalError: local variable 'a' referenced before assignment

>>>
>>>
>>> li = [1,]
>>> def f2():
...     li.append(1)
...     print li
...
>>> f2()
[1, 1]
>>> li
[1, 1]

```

## <5>总结2：

- 在函数中不使用global声明全局变量时不能修改全局变量的本质是不能修改全局变量的指向，即不能将全局变量指向新的数据。
- 对于不可变类型的全局变量来说，因其指向的数据不能修改，所以不使用global时无法修改全局变量。

- 对于可变类型的全局变量来说，因其指向的数据可以修改，所以不使用 global 时也可修改全局变量。

# 函数应用：学生管理系统

## 函数返回值(二)

在python中我们可不可以返回多个值？

```
>>> def divid(a, b):
...     shang = a//b
...     yushu = a%b
...     return shang, yushu
...
>>> sh, yu = divid(5, 2)
>>> sh
5
>>> yu
1
```

本质是利用了元组

# 函数参数(二)

## 1. 缺省参数

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。下例会打印默认的age，如果age没有被传入：

```
def printinfo( name, age = 35 ):  
    # 打印任何传入的字符串  
    print "Name: ", name  
    print "Age ", age  
  
# 调用printinfo函数  
printinfo(name="miki")  
printinfo( age=9, name="miki" )
```

以上实例输出结果：

```
Name: miki  
Age 35  
Name: miki  
Age 9
```

注意：带有默认值的参数一定要位于参数列表的最后面。

```
>>> def printinfo(name, age=35, sex):  
...     print name  
...  
File "<stdin>", line 1  
SyntaxError: non-default argument follows default argument
```

## 2. 不定长参数

有时可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，声明时不会命名。

基本语法如下：

```
def functionname([formal_args,] *args, **kwargs):
    "函数_文档字符串"
    function_suite
    return [expression]
```

加了星号 (\*) 的变量args会存放所有未命名的变量参数，args为元组；而加\*\*的变量kwargs会存放命名参数，即形如key=value的参数，kwargs为字典。

```
>>> def fun(a, b, *args, **kwargs):
...     """可变参数演示示例"""
...     print "a =", a
...     print "b =", b
...     print "args =", args
...     print "kwargs: "
...     for key, value in kwargs.items():
...         print key, "=", value
...
>>> fun(1, 2, 3, 4, 5, m=6, n=7, p=8) # 注意传递的参数对应
a = 1
b = 2
args = (3, 4, 5)
kwargs:
p = 8
m = 6
n = 7
>>>
>>>
>>>
>>> c = (3, 4, 5)
>>> d = {"m":6, "n":7, "p":8}
>>> fun(1, 2, *c, **d)      # 注意元组与字典的传参方式
```

```
a = 1
b = 2
args = (3, 4, 5)
kwargs:
p = 8
m = 6
n = 7
>>>
>>>
>>>
>>> fun(1, 2, c, d) # 注意不加星号与上面的区别
a = 1
b = 2
args = ((3, 4, 5), {'p': 8, 'm': 6, 'n': 7})
kwargs:
```

### 3. 引用传参

- 可变类型与不可变类型的变量分别作为函数参数时，会有什么不同吗？
- Python有没有类似C语言中的指针传参呢？

```
>>> def selfAdd(a):
...     """自增"""
...     a += a
...
>>> a_int = 1
>>> a_int
1
>>> selfAdd(a_int)
>>> a_int
1
>>> a_list = [1, 2]
>>> a_list
[1, 2]
>>> selfAdd(a_list)
>>> a_list
[1, 2, 1, 2]
```

Python中函数参数是引用传递（注意不是值传递）。对于不可变类型，因变量不能修改，所以运算不会影响到变量自身；而对于可变类型来说，函数体中的运算有可能会更改传入的参数变量。

## 想一想为什么

```
>>> def selfAdd(a):
...     """自增"""
...     a = a + a    # 我们更改了函数体的这句话
...
>>> a_int = 1
>>> a_int
1
>>> selfAdd(a_int)
>>> a_int
1
>>> a_list = [1, 2]
>>> a_list
[1, 2]
>>> selfAdd(a_list)
>>> a_list
[1, 2]      # 想一想为什么没有变呢?
```

# 递归函数

## <1>什么是递归函数

通过前面的学习知道一个函数可以调用其他函数。

如果一个函数在内部不调用其它的函数，而是自己本身的话，这个函数就是递归函数。

## <2>递归函数的作用

举个例子，我们来计算阶乘  $n! = 1 * 2 * 3 * \dots * n$

解决办法1：

```
/test.py - Sublime Text
test.py
1 #coding=utf-8
2
3 def calNum(num):
4     i = 1
5     result = 1
6
7     while i<=num:
8         result *= i
9         i+=1
10
11     return result
12
13 ret = calNum(3)
14 print(ret)
15
16
```

使用循环来完成

```
python@ubuntu:~/Desktop$ python3 test.py
6
python@ubuntu:~/Desktop$
```

看阶乘的规律

```
1! = 1
2! = 2 × 1 = 2 × 1!
3! = 3 × 2 × 1 = 3 × 2!
4! = 4 × 3 × 2 × 1 = 4 × 3!
...
n! = n × (n-1)!
```

## 解决办法2:

```
/test.py - Sublime Text
test.py

1 #coding=utf-8
2
3 def calNum(num):
4     if num>=1:
5         result = num * calNum(num-1)
6     else:
7         result = 1
8     return result
9
10 # def calNum(num):
11 #     i = 1
12 #     result = 1
13
14 #     while i<=num:
15 #         result *= i
16 #         i+=1
17
18 #     return result
19
20 ret = calNum(3)
21 print(ret)
```

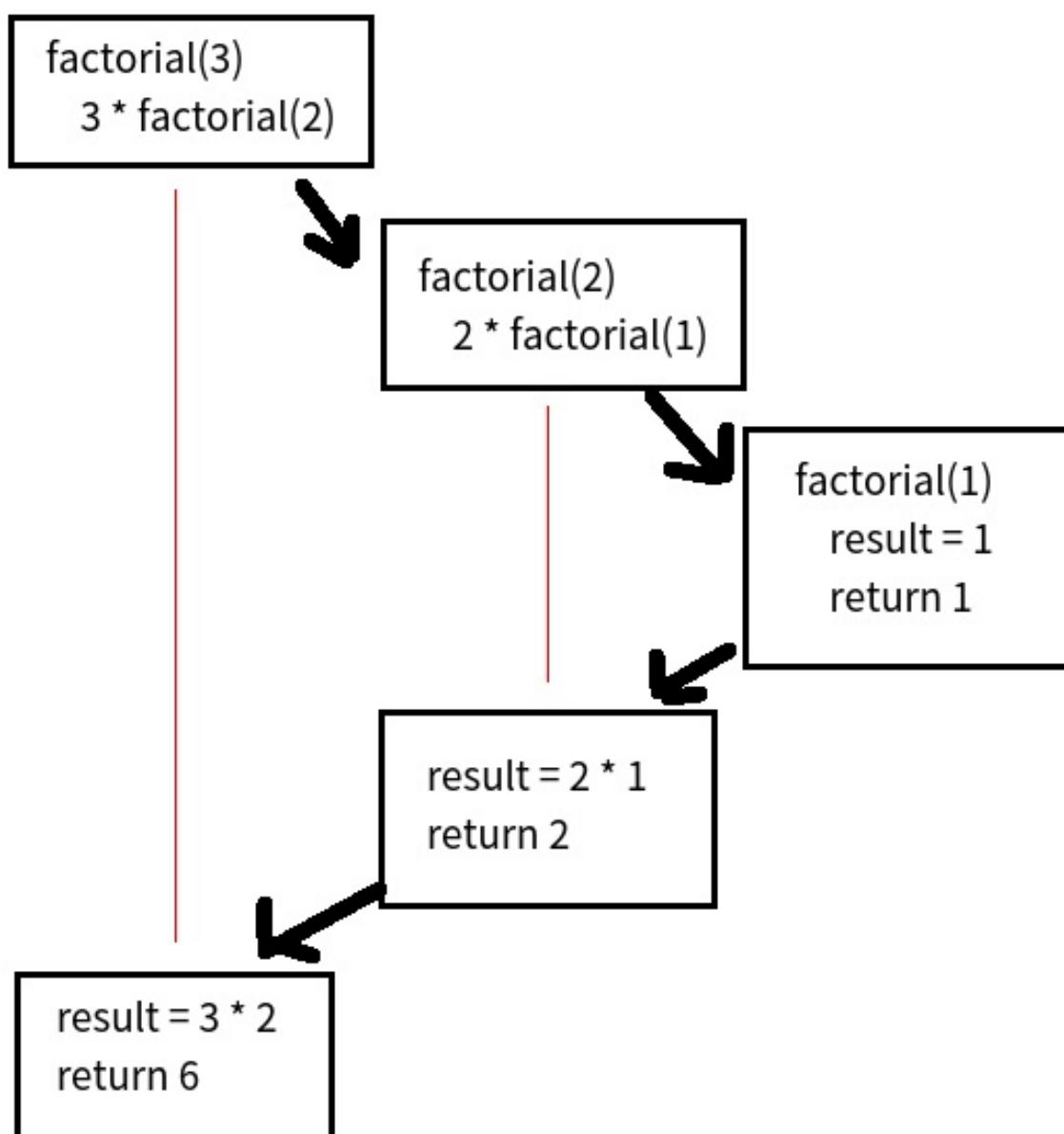
使用递归来完成

```
python@ubuntu: ~/Desktop
python@ubuntu:~/Desktop$ python3 test.py
6
python@ubuntu:~/Desktop$
```

## 原理

```
def factorial(num):  
    if num>1:  
        result = num * factorial(num-1)  
    else:  
        result = 1  
    return result
```

factorial(3)调用过程：





# 匿名函数

用lambda关键词能创建小型匿名函数。这种函数得名于省略了用def声明函数的标准步骤。

lambda函数的语法只包含一个语句，如下：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

如下实例：

```
sum = lambda arg1, arg2: arg1 + arg2

#调用sum函数
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

以上实例输出结果：

```
Value of total : 30
Value of total : 40
```

Lambda函数能接收任何数量的参数但只能返回一个表达式的值

匿名函数不能直接调用print，因为lambda需要一个表达式

## 应用场合

### 函数作为参数传递

#### 1. 自己定义函数

```
>>> def fun(a, b, opt):
...     print "a =", a
...     print "b =", b
...     print "result =", opt(a, b)
...
>>> fun(1, 2, lambda x,y:x+y)
a = 1
b = 2
result = 3
```

## 2. 作为内置函数的参数

想一想，下面的数据如何指定按age或name排序？

```
stus = [
    {"name": "zhangsan", "age": 18},
    {"name": "lisi", "age": 19},
    {"name": "wangwu", "age": 17}
]
```

按name排序：

```
>>> stus.sort(key = lambda x:x['name'])
>>> stus
[{'age': 19, 'name': 'lisi'}, {'age': 17, 'name': 'wangwu'}, {'age': 18, 'name': 'zhangsan'}]
```

按age排序：

```
>>> stus.sort(key = lambda x:x['age'])
>>> stus
[{'age': 17, 'name': 'wangwu'}, {'age': 18, 'name': 'zhangsan'}, {'age': 19, 'name': 'lisi'}]
```



# 函数使用注意事项

## 1. 自定义函数

### <1>无参数、无返回值

```
def 函数名():
    语句
```

### <2>无参数、有返回值

```
def 函数名():
    语句
    return 需要返回的数值
```

注意：

- 一个函数到底有没有返回值，就看有没有return，因为只有return才可以返回数据
- 在开发中往往根据需求来设计函数需不需要返回值
- 函数中，可以有多个return语句，但是只要执行到一个return语句，那么就意味着这个函数的调用完成

### <3>有参数、无返回值

```
def 函数名(形参列表):
    语句
```

注意：

- 在调用函数时，如果需要把一些数据一起传递过去，被调用函数就需要用参数来接收

- 参数列表中变量的个数根据实际传递的数据的多少来确定

## <4>有参数、有返回值

```
def 函数名(形参列表):
    语句
    return 需要返回的数值
```

## <5>函数名不能重复

```
test.py
1 #coding=utf-8
2
3 def test():
4     print("你好啊")
5
6 def test(a,b):
7     print("你好啊ab")
8
9 test()
10
```

dongGe@bogon Desktop\$ python test.py  
Traceback (most recent call last):  
File "test.py", line 9, in <module>  
 test()  
TypeError: test() takes exactly 2 arguments (0 given)  
dongGe@bogon Desktop\$

函数的名字不要相同

## 2. 调用函数

### <1>调用的方式为：

```
函数名([实参列表])
```

### <2>调用时，到底写不写 实参

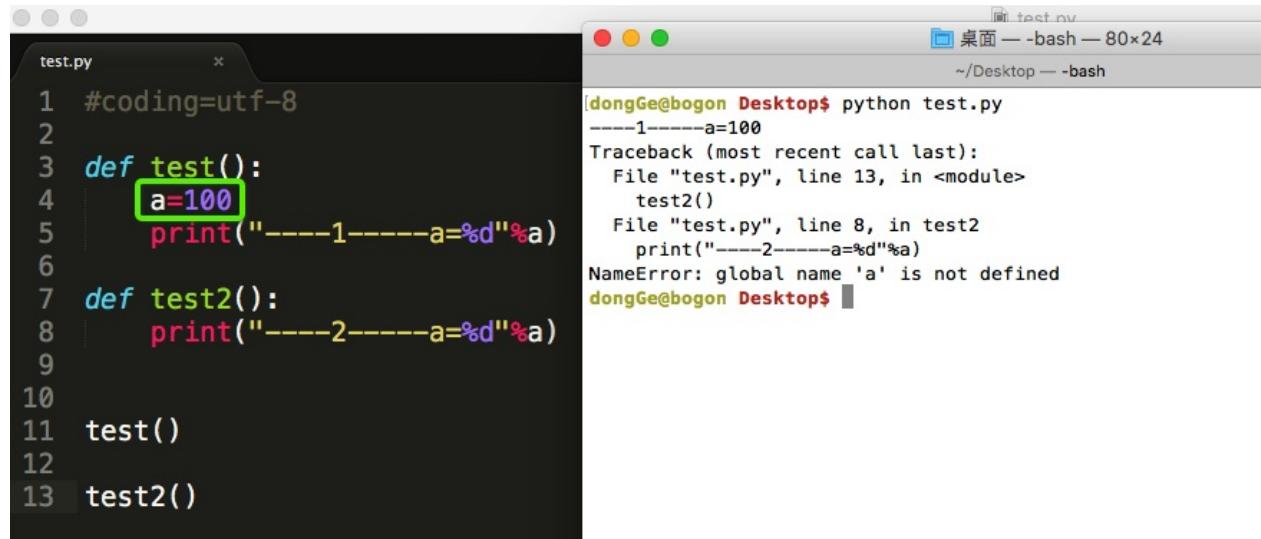
- 如果调用的函数 在定义时有形参，那么在调用的时候就应该传递参数

### <3>调用时，实参的个数和先后顺序应该和定义函数中要求的一致

### <4>如果调用的函数有返回值，那么就可以用一个变量来进行保存这个值

### 3. 作用域

<1>在一个函数中定义的变量，只能在本函数中用(局部变量)



```
test.py
1 #coding=utf-8
2
3 def test():
4     a=100
5     print("----1----a=%d"%a)
6
7 def test2():
8     print("----2----a=%d"%a)
9
10 test()
11
12 test2()

[dongGe@bogon Desktop]$ python test.py
----1----a=100
Traceback (most recent call last):
  File "test.py", line 13, in <module>
    test2()
  File "test.py", line 8, in test2
    print("----2----a=%d"%a)
NameError: global name 'a' is not defined
[dongGe@bogon Desktop]$
```

<2>在函数外定义的变量，可以在所有的函数中使用(全局变量)

# 作业

## 必做题

### 1. 编程实现 9\*9乘法表

提示：使用循环嵌套

### 2.用函数实现求100-200里面所有的素数

提示：素数的特征是除了1和其本身能被整除，其它数都不能被整除的数

### 3.请用函数实现一个判断用户输入的年份是否是闰年的程序

提示：

1. 能被400整除的年份
  2. 能被4整除，但是不能被100整除的年份
- 以上2种方法满足一种即为闰年

## 选做题

### 1.用函数实现输入某年某月某日，判断这一天是这一年的第几天？闰年情况也考虑进去

20160818  
是今年第x天

## 2. 编写“学生管理系统”，要求如下：

必须使用自定义函数，完成对程序的模块化

学生信息至少包含：姓名、年龄、学号，除此以外可以适当添加

必须完成的功能：添加、删除、修改、查询、退出