# 1. 函数介绍

## 1.1. 打印99乘法表

请看如下代码:

i = 1

while i <= 9 :

j = 1

while j <= i :

print("%d\*%d=%d\t" % (j,i,i\*j),end="")

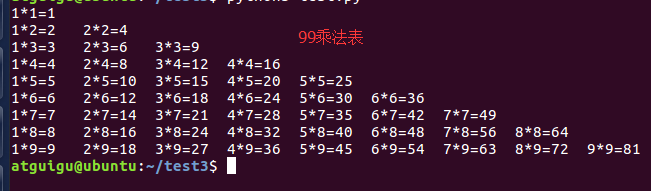
j += 1

i += 1

#print("\n",end="")

print("")

运行后的现象:



## 1.2. 痛点引入

需求一：如果一个程序在不同的地方需要输出“99乘法表”，程序应该怎样设计？

if 条件1:

输出‘99乘法表’

...(省略)...

if 条件2:

输出‘99乘法表’

...(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？

需求二：打印100次99乘法表

## 1.3. 把打印99乘法表封装成一个函数

#定义一个名字叫print\_99的函数

def print\_99():

i = 1

while i <= 9 :

j = 1

while j <= i :

print("%d\*%d=%d\t" % (j,i,i\*j),end="")

j += 1

i += 1

#print("\n",end="")

print("")

#调用函数

print\_99()

## 1.4. 用函数解决问题

现在来实现打印100次99乘法表

## 1.5. 函数的好处

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**

# 2. 函数定义

## 2.1. 定义函数

定义函数的格式如下：

def 函数名():

代码

**函数简单使用**

# 定义一个函数，能够完成打印信息的功能

def print\_info():

print('------------------------------------')

print( ' 人生苦短，我用Python')

print( '------------------------------------')

#调用函数

print\_info()

**注意：**

标识符由字母、下划线和数字组成，且数字不能开头

Python推荐:函数名应该为小写，可以用下划线风格单词以增加可读性。

## 2.2. 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

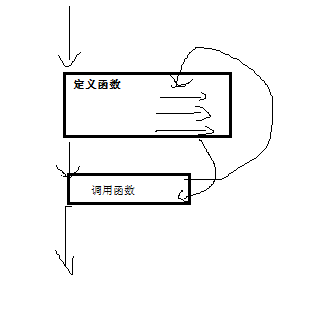
print\_info()

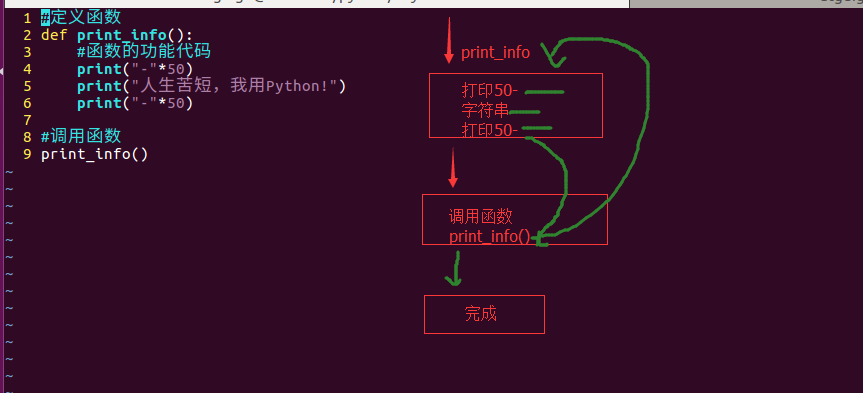
## 2.3. 分析函数执行的过程

### 2.3.1. 回顾程序执行的三种流程

顺序执行，选择执行，循环执行

### 2.3.2. 函数执行的流程





## 2.4. 练一练-使用函数打印自己的信息

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

提示：使用def定义函数，编写完函数之后，通过 函数名() 进行调用。

函数命名要见明知意

# 3. 有传参有返回值的函数

## 3.1. 需求

用户输入输入两个数字让其相加，并且把结果返回，接收后打印出来

## 3.2. 定义带有两个参数的函数

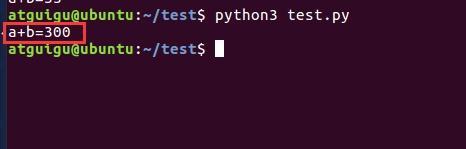
#定义函数

def add\_num(a,b):

print("a+b=%d"%(a+b))

#调用函数

add\_num(100,200)



## 3.3. 调用函数

#调用函数

add\_num(100,200)

## 3.4. 保存函数的返回值并且打印

#定义函数

def add\_num(a,b):

#把结果相加

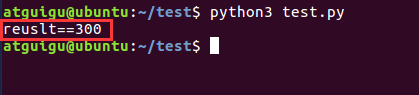
add = a + b

return add

#调用函数

result = add\_num(100,200)

print("reuslt==%d" % result )



# 4. 函数的类型

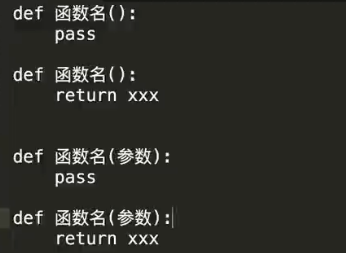
函数根据有没有参数，有没有返回值，可以相互组合，一共有4种:

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



## 4.1. 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，打印提示等类似的功能，使用这类的函数

#打印软件菜单信息

def print\_menu():

print('-'\*50)

print(' 名片管理系统1.0')

print('')

print(' 1. 添加名片')

print(' 2. 删除名片')

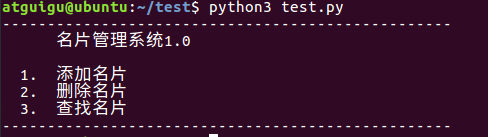
print(' 3. 查找名片')

print('-'\*50)

#调用函数

print\_menu()

结果:



## 4.2. 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 get\_temperature():

#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 = get\_temperature()

print('当前的温度为:%d'%temperature)

结果:



## 4.3. 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

# 0关灯，1开灯

def set\_on\_off(isOffOn):

if isOffOn == 0 :

print("关灯")

elif isOffOn == 1 :

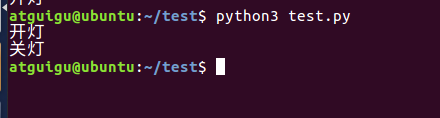
print("开灯")

#调用函数

set\_on\_off(1)

set\_on\_off(0)

运行结果



## **4.4. 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

### 4.4.1. 计算累计合1~？

# 计算1~?的累积和

def calc\_num(num):

result = 0

i = 1

while i<=num:

result = result + i

i+=1

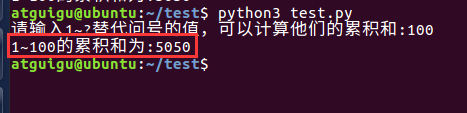
return result

num = int(input("请输入1~?替代问号的值，可以计算他们的累积和:"))

result = calc\_num(num)

print('1~%d的累积和为:%d'% (num,result))

结果:



## 4.5. 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

# 5. 给函数添加的文档说明

## 5.1. 给函数添加文档说明

>>> def add(a,b):

... "用来完成对2个数求和"

... print("%d"%(a+b))...

>>>

>>> add(11,22)

33

说明：给函数添加说明可以使用单双引用和双单三引用（共计4种）都可以

如果执行，以下代码

>>> help(add)

## 5.2. 能够看到函数的相关说明

Help on function add in module \_\_main\_\_:

add (a, b)

用来完成对2个数求和

(END)

输入q退出

## 5.3. 工作中的生成文档

### 5.3.1. 生成文档

文件名字叫test.py

def initData():

'''

初始化商品的价格、日期、分类等

'''

pass

def upDateIt():

'''

用来更新商品信息

'''

pass

def findIt():

'''

查找商品信息

'''

pass

def deleteIt():

'''

删除过期下架商品信息

'''

pass

def createIt():

'''

创建新商品及上架信息

'''

pass

if \_\_name\_\_ == '\_\_main\_\_':

import test

print(help(test))

### 5.3.2. 文档效果



# 6. 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 add\_num():

a = 11

b = 22

c = a+b

print(c)

add\_num()

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

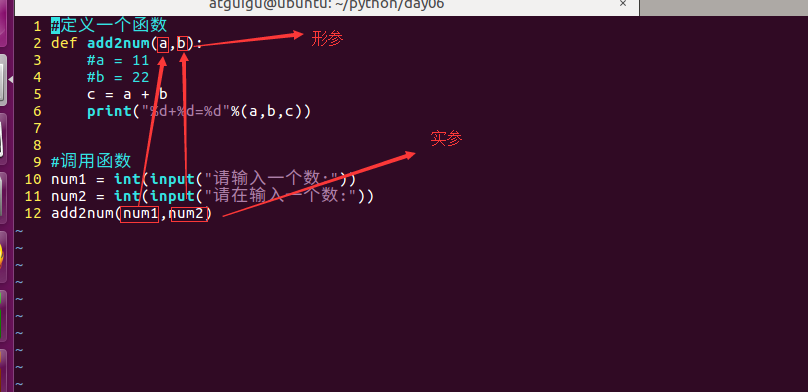
## 6.1. 定义带有参数的函数

示例如下：

def add\_2\_num(a, b):

c = a + b

print(c)



## 6.2. 调用带有参数的函数

以调用上面的add\_2\_num(a, b)函数为例:

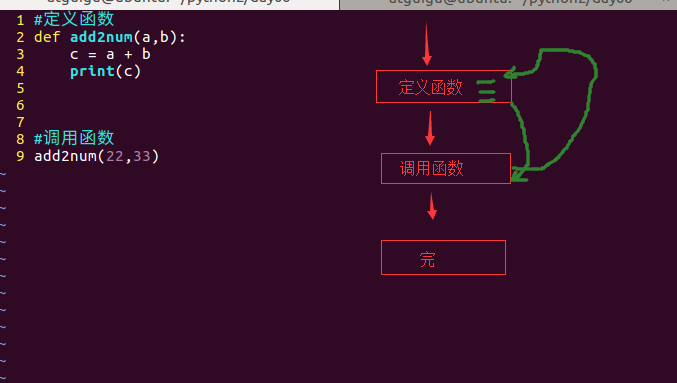
def add\_2\_num(a, b):

c = a + b

print(c)

add\_2\_num(100,200) #调用带有参数的函数时，需要在小括号中，传递数据

### 6.2.1. 调用带有参数函数的运行过程



## 6.3. 调用函数时参数的顺序

>>> def add\_2\_num(a,b):

... print(a,b)...

>>> add\_2\_num(1,2)

1 2

>>> add\_2\_num**(b=1,a=2)**

2 1

>>>

>>>

add\_2\_num(b=1,2)

File "<stdin>", line 1

SyntaxError: positional argument follows keyword argument

>>>

>>>

## 6.4. 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”

## **6.5. 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。

# **7. 函数参数(二)**

## 7.1. 缺省参数

### 7.1.1. 缺省参数简介

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：

#-\*-coding:utf-8-\*-

#定义函数

def print\_info(name, age = 35 ):

# 打印任何传入的字符串

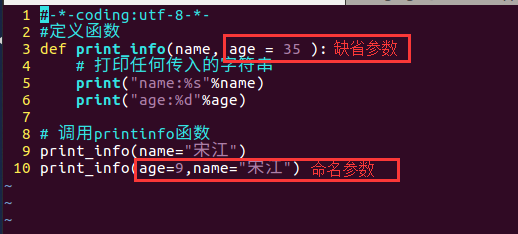
print("name:%s"%name)

print("age:%d"%age)

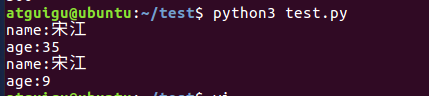
# 调用printinfo函数

print\_info(name="宋江")

print\_info(age=9,name="宋江")



以上实例输出结果：



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

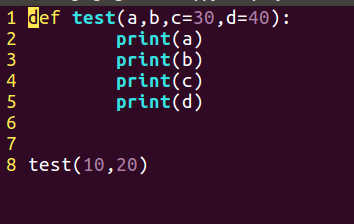
...

File "<stdin>", line 1

SyntaxError: non-default argument follows default argument

### 7.1.2. 缺省参数的其他特点

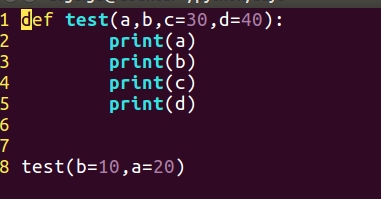
**案例1：**



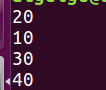
输出结果



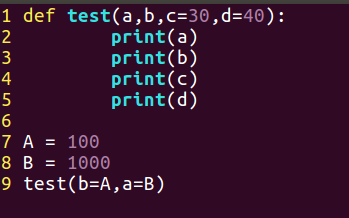
**案例2：**



输出结果



**案例3:**



输出结果



## 7.2. 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):

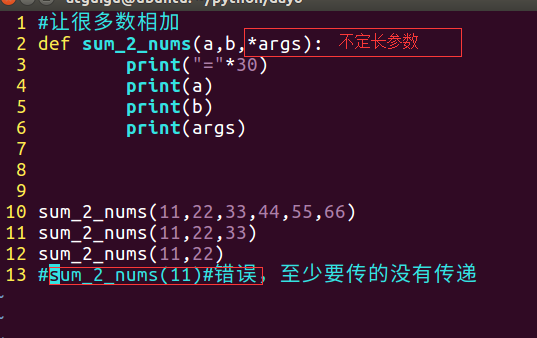
"函数\_文档字符串"

function\_suite

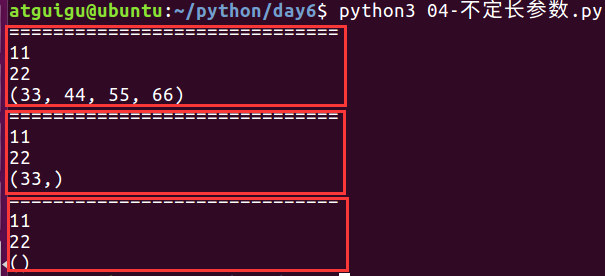
return [expression]

### 7.2.1. 不定参数\*args

#### 7.2.1.1. 应用场景-让多个数做运算

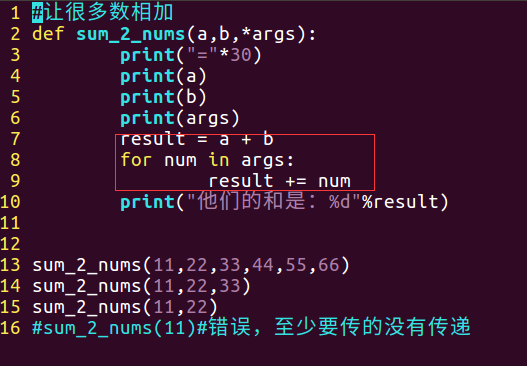


输入结果如图：

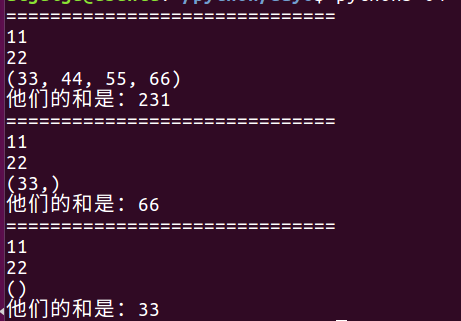


#### 7.2.1.2. 让多个数做加法运算

案例代码

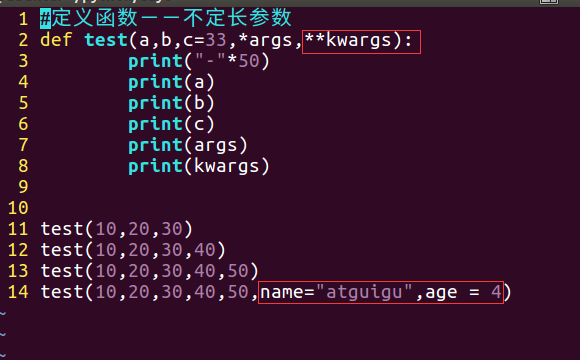


输出结果

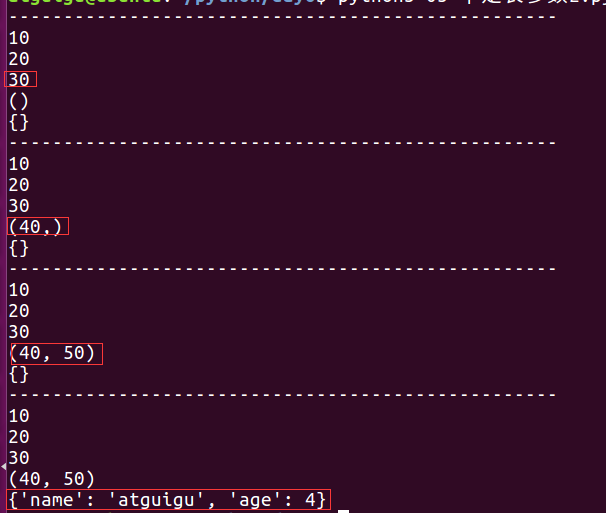


### 7.2.2. 不定长参数\*\*kwargs

#### 7.2.2.1. 案例代码

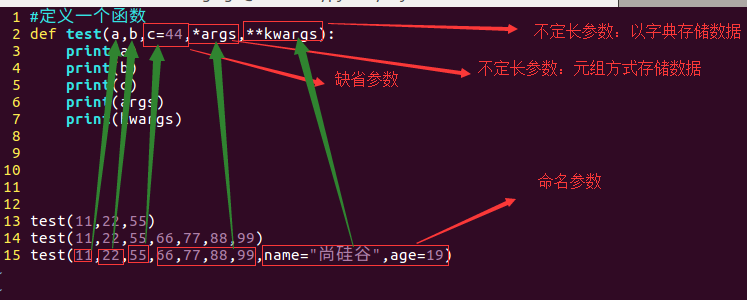


输出结果

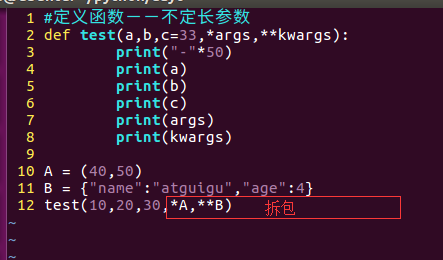


### 7.2.2. 小结

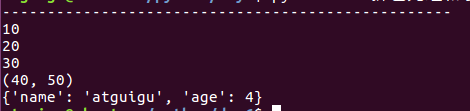
加了星号（\*）的变量args会存放所有未命名的变量参数，args为元组；而加\*\*的变量kwargs会存放命名参数，即形如key=value的参数， kwargs为字典。



## 7.3. 拆包



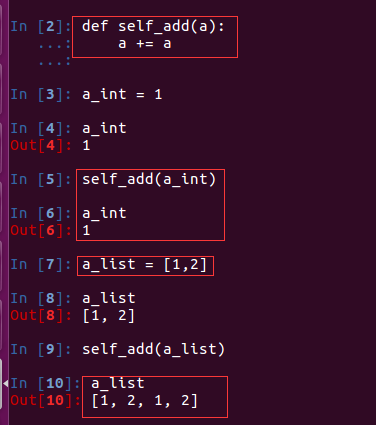
运行结果

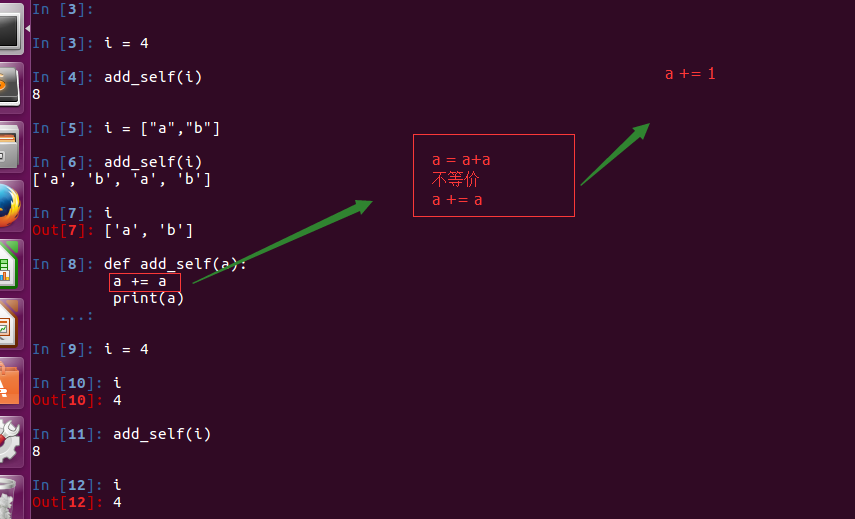


## 7.4. 引用传参

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)

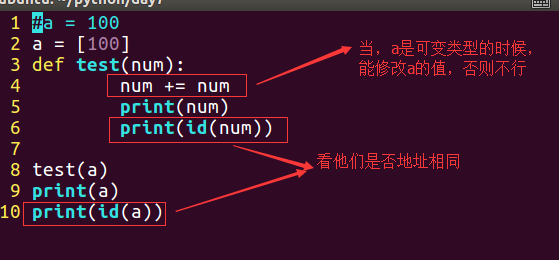
>>> a\_list

[1, 2] # 想一想为什么没有变呢？

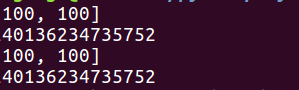
## 7.5. 扩展--在Python中a=a+a并不等价a +=a

### 7.5.1. a +=a方式

这种方式还是使用原来的地址

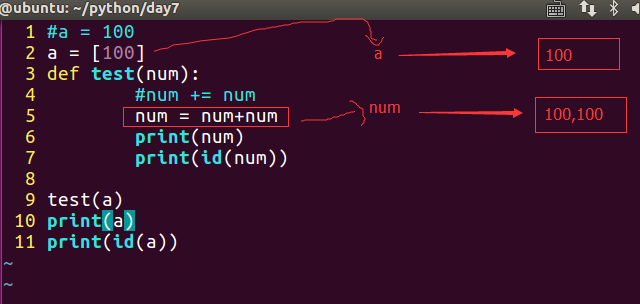


运行结果



### 7.5.2. . a = a+a方式

这种方式开辟了新的空间



被重新引用了，打印输出内容是：

# 8. 函数返回值

## 8.1. 返回值介绍

现实生活中的场景:

我给儿子10块钱，让他给我打酱油。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，让儿子打酱油这个事情最终的目标是，让他把酱油给带回来，然后给到手上，此时酱油就是返回值

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。

## 8.2. 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 add\_2\_num(a, b):

c = a+b

return c

或者

def add\_2\_num(a, b):

return a+b

## 8.3. 保存函数的返回值

在本小节刚开始的时候，说过的“打酱油”的例子中，最后儿子给你酱油时，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 add\_2\_num(a, b):

return a+b

#调用函数，顺便保存函数的返回值

result = add\_2\_num(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:



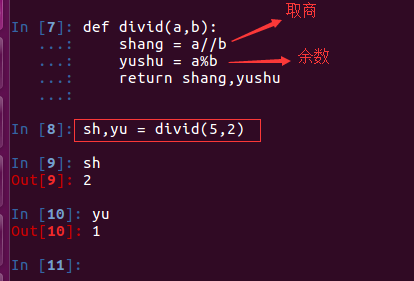
## 8.5. 函数返回多个值

参考代码



执行多次的结果





# 9. 函数的嵌套调用

## 9.1. 函数调用函数

### 9.1.1. 案例

def testB():

print('---- testB start----')

print('这里是testB函数执行的代码...(省略)...')

print('---- testB end----')

def testA():

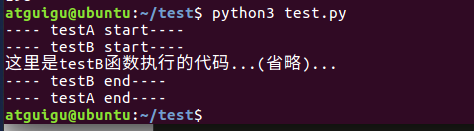
print('---- testA start----')

testB()

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

testA()

结果：

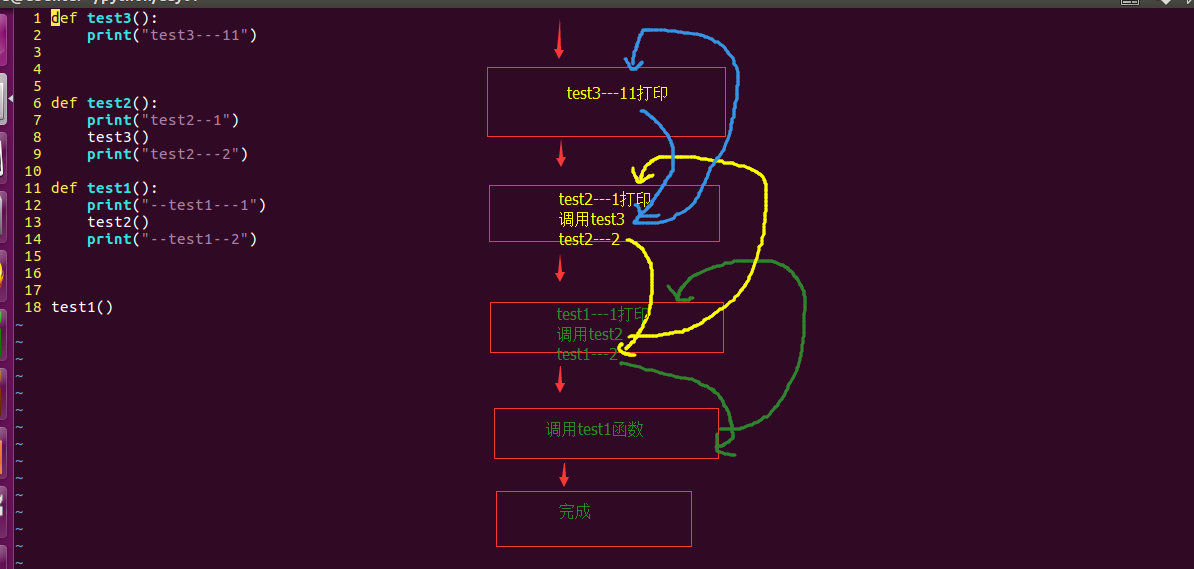


### 9.1.2. 小总结

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**

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

画图分析



# 10. 函数嵌套应用：打印图形和数学计算

## 10.1. 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

## 10.2. 思考-打印一条线和打印多行线

写一个函数打印一条横线

打印自定义行数的横线

参考代码

# 打印一条横线

def print\_one\_line():

print("-"\*30)

# 打印多条横线

def print\_num\_line(num):

i=0

# 因为print\_one\_line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:

print\_one\_line()

i += 1

print\_num\_line(3)

## **10.3. 思考-求三个数的和和三个数的平均值**

写一个函数求三个数的和

写一个函数求三个数的平均值

**参考代码**

# 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)

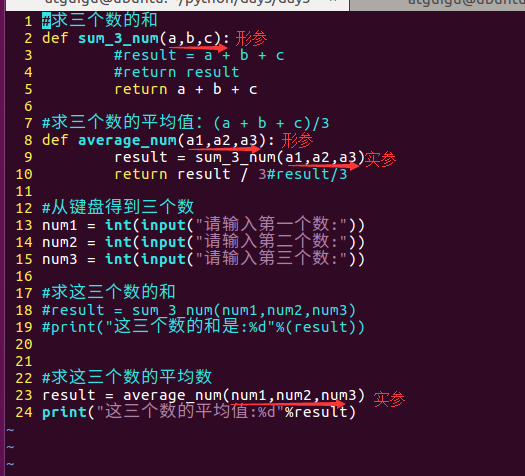
aveResult = sumResult/3.0

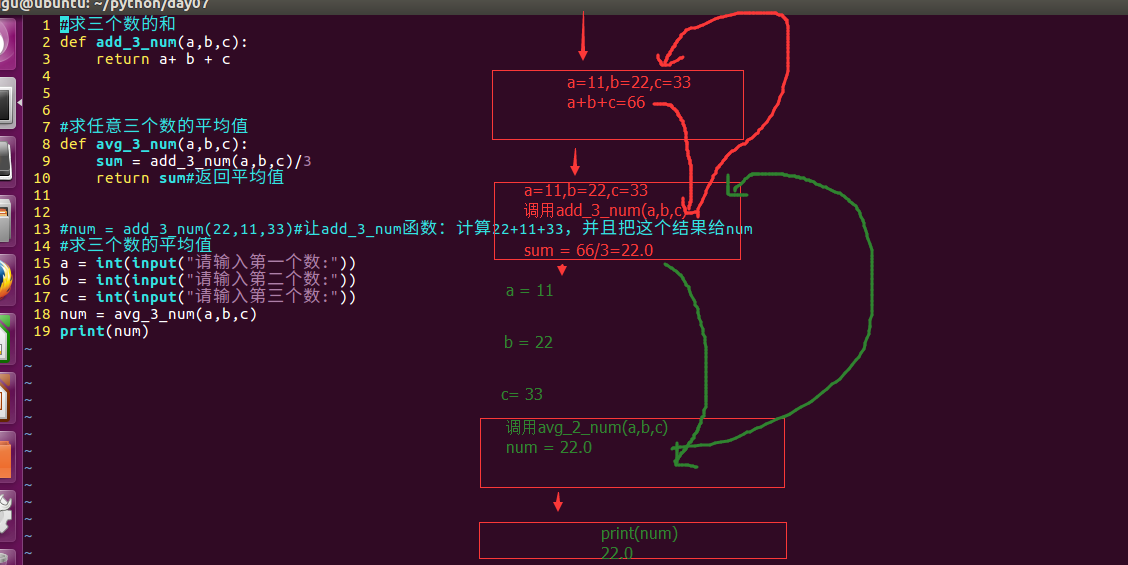
return aveResult

# 调用函数，完成对3个数求平均值

result = average3Number(11,2,55)

print("average is %d"%result)





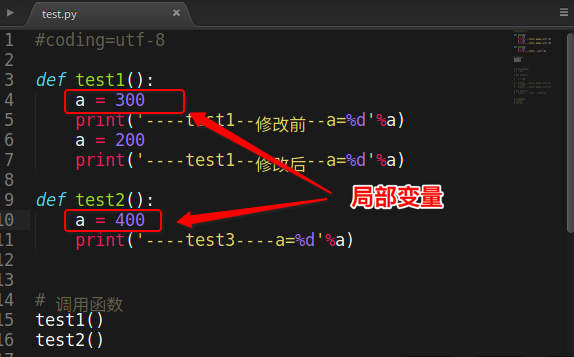
# 11. 局部变量

**局部变量，就是在函数内部定义的变量**

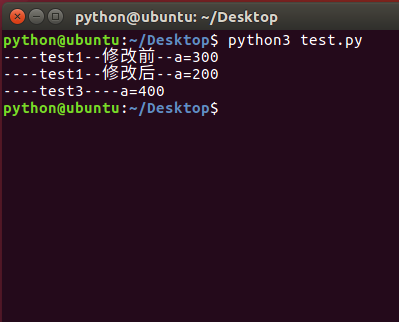
不同的函数，可以定义相同的名字的局部变量，但是各用个的不会产生影响.

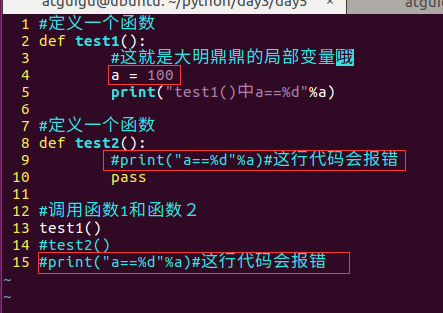
## 11.1. 什么是局部变量

如下图所示:



运算结果





## 11.2. 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

# **12. 全局变量**

## 12.1. 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()

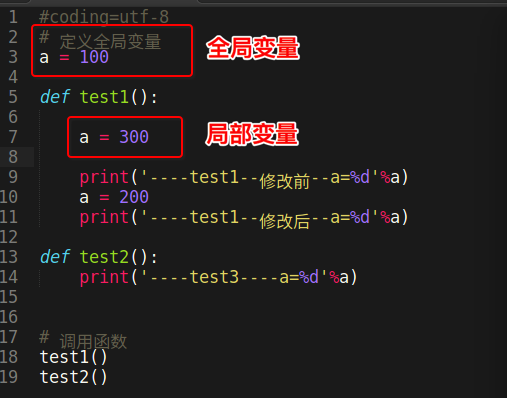
运行结果:

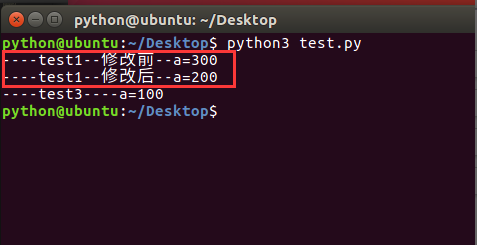
01-第7天-7

## 12.2. 全局变量和局部变量名字相同问题

看如下代码:

全局变量和局部变量同名的时候，函数使用的是局部变量





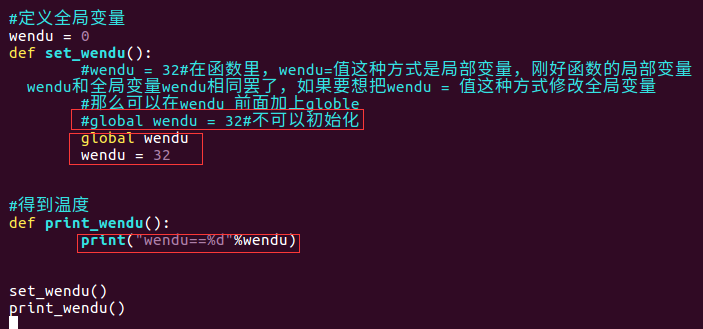
## 12.3. 在函数值添加global后修改全局变量

既然全局变量，就是能够在所以的函数中进行使用，那么可否进行修改呢？

**代码如下:**







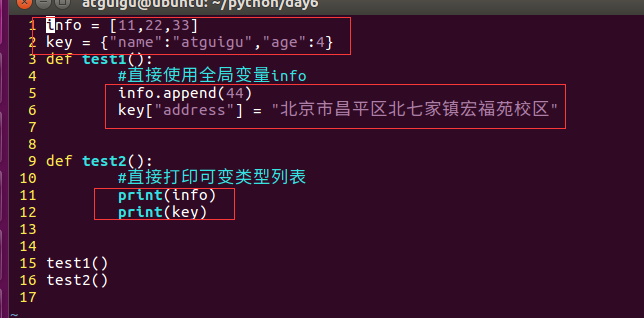
**全局变量命名建议**

建议前面加上个g

例如g\_wendu = 32#全局变量

## 12.4. 可变类型的全局变量

如果把可变类型当全局变量，在函数里使用，不用加global声明，如果加不会报错。



>>> 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

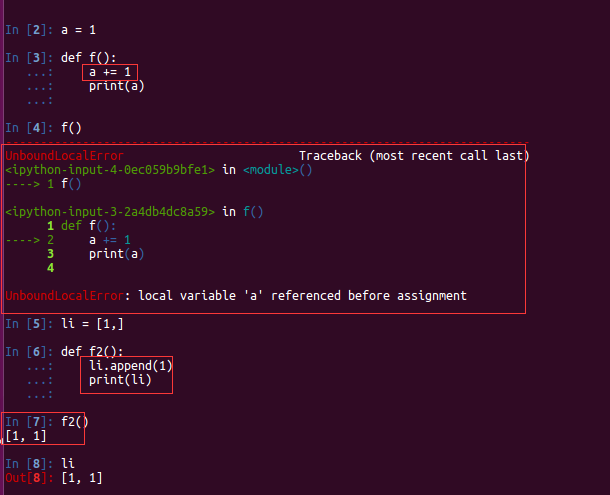
...

>>> f2()

[1, 1]

>>> li

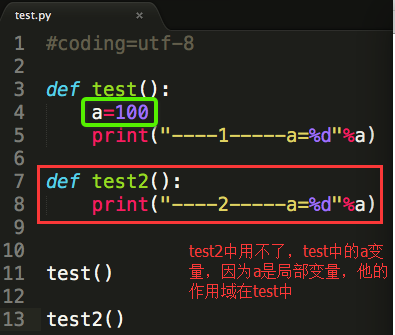
[1, 1]



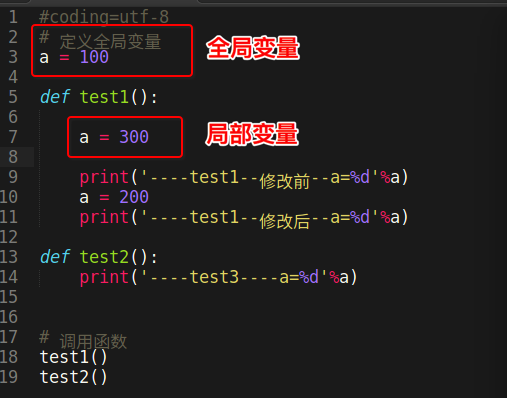
## 12.5. 局部变量和全局变量的作用域

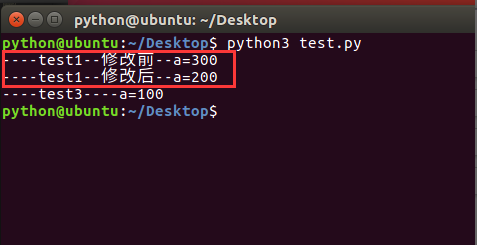
作用域

### 12.5.1. 在一个函数中定义的变量，只能在本函数中用(局部变量)



### 12.5.2. 在函数外定义的变量，可以在所有的函数中使用(全局变量)





## 12.6. 总结

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**

**全局变量要定义在函数调用前，否则会报错**

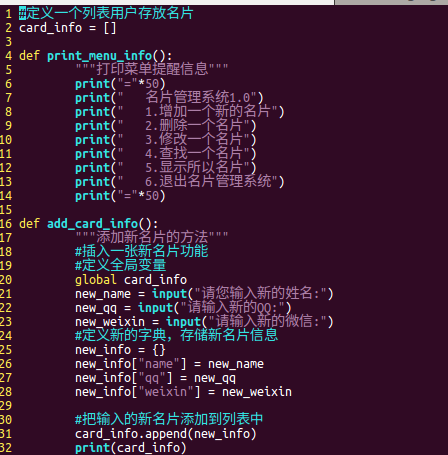
在函数中不使用global声明全局变量时不能修改全局变量的本质是不能修改全局变量的指向，即不能将全局变量指向新的数据。

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。

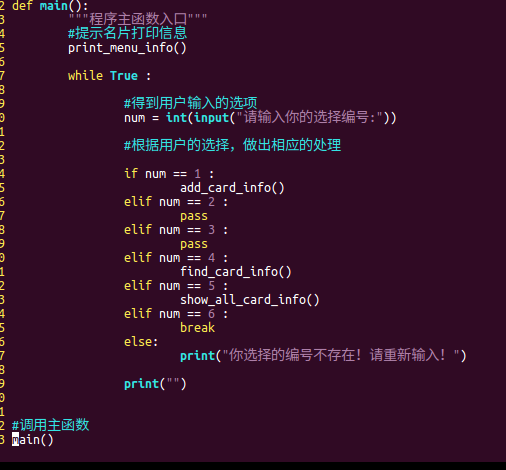
**对于可变类型的全局变量来说，因其指向的数据可以修改，所以不使用global时也可修改全局变量。**

# **13. 函数应用：名片管理系统**

## 13.1. 参考代码如图







## 13.2. py文件中写代码顺序

#!/usr/bin/python

utf-8

import xxx

全局变量

函数的定义

定义类

---------下面都是调用-------

调用函数

创建类的实例对象

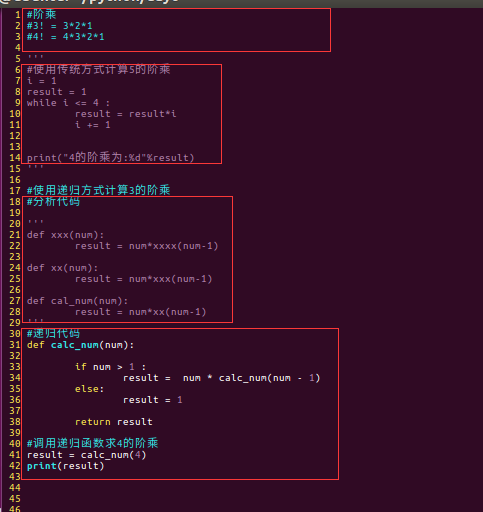
执行实例对象的函数

# 13. 递归函数

## 13.1. 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**。



#递归函数：特殊的函数嵌套，函数调用本身自己

#4! = 4\*3!

#3! = 3\*2!

#2! = 2\*1

#本质

#4! = 4\*3\*2\*1=24

#一、使用while循环实现计算阶乘

i = 1

sum = 1

while i <= 4 :

sum \*= i

i += 1

print("使用while循环计算4的阶乘==%d" % sum)

#二、使用函数嵌套实现计算阶乘

def x(num):

if num == 1 :

return num

def xx(num):

#当前num=2,计算2\*1!!

return 2\*x(num-1)

def xxx(num):

#当前num=3,计算3\*2!

return 3\*xx(num-1)

def calc\_num(num):

#当前num = 4，计算4\*3!

return 4\*xxx(num-1)

print("4的阶乘是==%d" % calc\_num(4))

#三、使用递归函数实现计算阶乘

def calc\_num(num):

#当前num = 4，计算4\*3!

if num <= 1:

return 1

return num\*calc\_num(num-1)

#调用递归函数

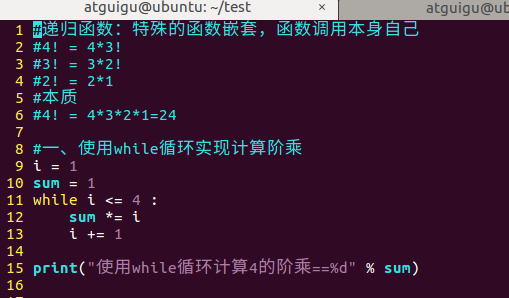
a = calc\_num(4)

print("递归函数计算4的阶乘==%d" % a)

## 13.2. 递归函数的理解

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

### 13.2.1. 使用while循环计算阶乘



看阶乘的规律

1! = 1

2! = 2 × 1 = 2 × 1!

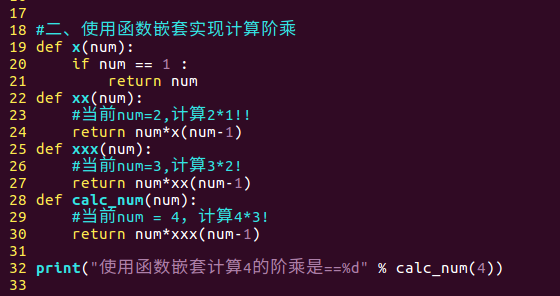
3! = 3 × 2 × 1 = 3 × 2!

4! = 4 × 3 × 2 × 1 = 4 × 3!

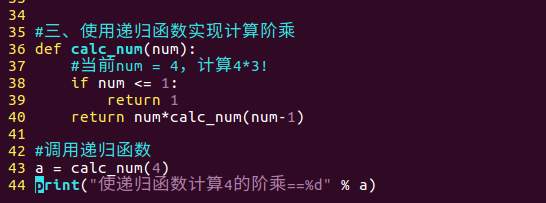
...

n! = n × (n-1)!

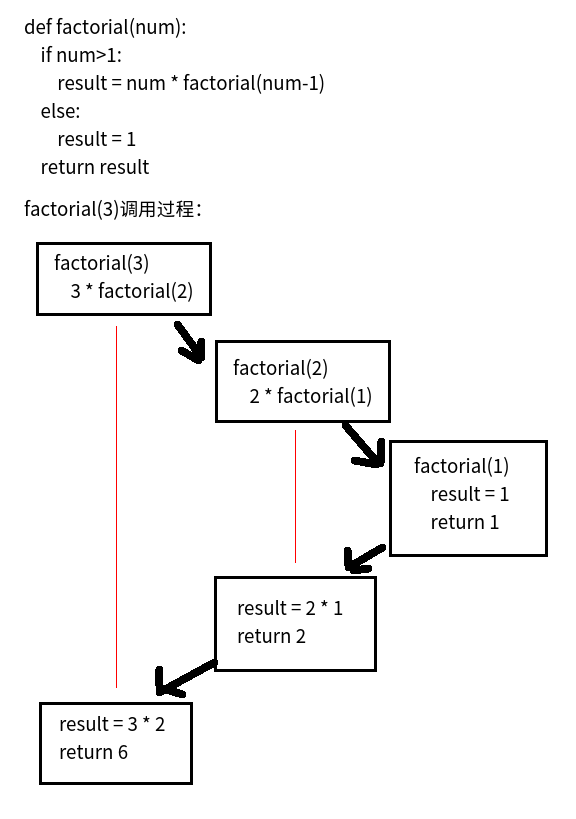
### 13.2.2. 使用函数嵌套计算阶乘



### 13.2.2. 使用递归计算阶乘



递归原理



## 13.3.递归扩展和小结

递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出

使用递归函数的优点是逻辑简单清晰，缺点是过深的调用会导致栈溢出。

# 15. 匿名函数

## 15.1. 匿名函数定义

**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

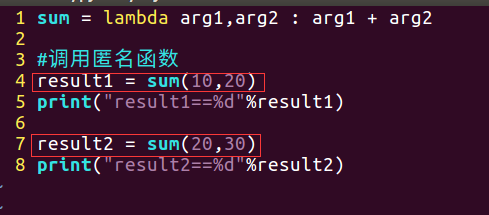
print("Value of total : %"%sum( 10, 20 ))

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**

## **15.2. 应用场合**

### 15.2.1. 函数作为参数传递

自己定义函数

>>> def fun(a, b, opt):...

print("a =%d"%a)...

print("b =%d"%b)...

print("result =%d"%opt(a, b))

...>>> fun(1, 2, lambda x,y:x+y)

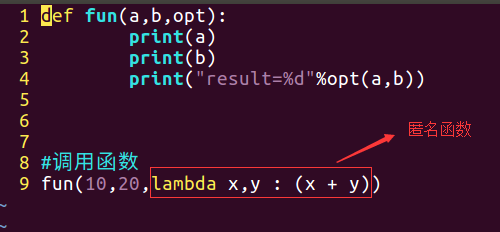
a = 1

b = 2

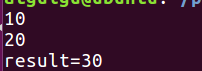
result = 3

#### 15.2.1.1. 作为内置函数的参数-表达式

代码



运行结果



#### 1.5.2.1.2. 作为内置参数传递-排序

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},

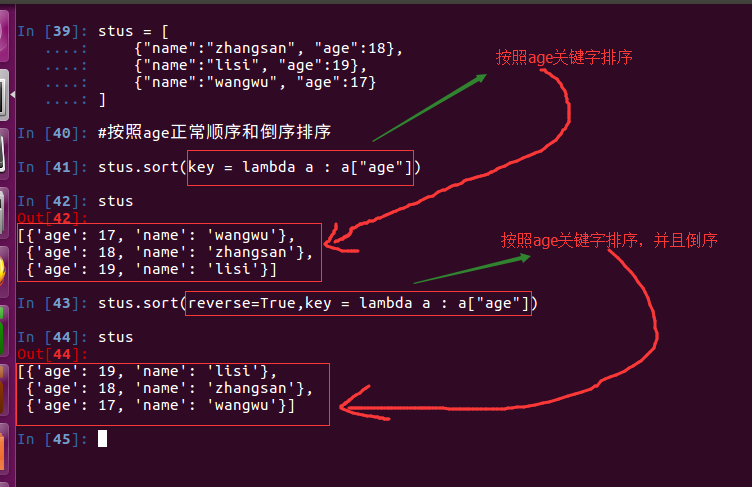
{"name":"wangwu", "age":17}

]

按name排序：

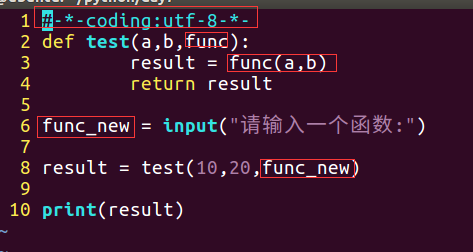


按age排序：

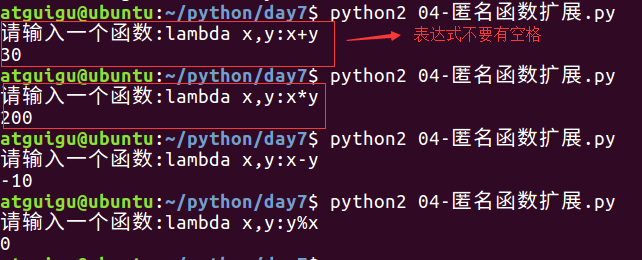


## 15.3. 匿名函数的扩展-彰显动态语言特点

### 15.3.1. 在Python2的写法：

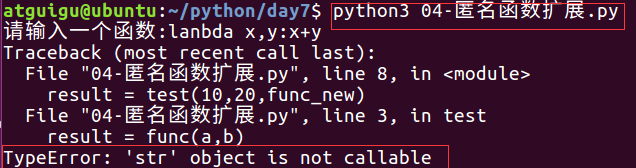


执行代码：

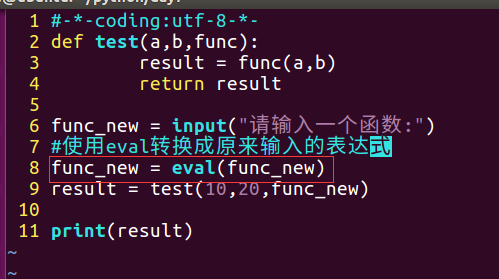


### 15.3.2. 在Python3的写法：

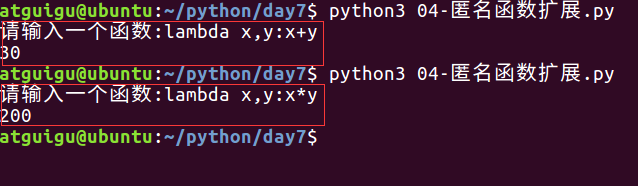
直接用Python3会报错



#### 修改成Python3支持的代码-eval



执行代码：



# **16. 函数的总结和注意事项**

## 16.1. 自定义函数

### 16.1.1. 无参数、无返回值

def 函数名():

语句

### 16.1.2. 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

### 16.1.3. 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

### 16.1.4. 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

### 16.1.5. 函数名不能重复



**如果重复了，上面的声明失效，只能用下面的了**

## 16.2. 调用函数

### 16.2.1. 调用的方式为：

函数名([实参列表])

### 16.2.2. 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

### 16.2.3. 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

### 16.2.4. 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

# **18. 总结与作业**

## 必做题

### 1. 编程实现 9\*9乘法表-函数版

提示：使用while循环嵌套,和函数嵌套

### 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

提示：

101 103 107 109 113   
127 131 137 139 149   
151 157 163 167 173   
179 181 191 193 197   
199

### 3.Python中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

提示下面的是闰年：

1696，1704,1708,1712,1716,1720,1724,1728,1732,1736,1740,1744,1748,1752,1756,1760,1764,1768,1772,1776,1780,1784,1788,1792,1798,1804,1808,1812,1816,1820,1824,1828,1832,1836,1840,1844,1848,1852,1856,1860,1864,1868,1872，1876,1880，1884,1888,1892,1896,1904,1908,1912,1916,1920,1924,1928,1932,1936,1940,1944,1948,1952,1956,1960,1964,1968,1972,1976,1980,1984,1988,1992,1996,2000,2004,2008,2012,2016,2020,2024,2028,2032.

### 4. 编写“名片管理系统”，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、地址，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

## 选做题

### 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

参考：https://zhidao.baidu.com/question/1115326546601604979.html

20160818

是今年第x天