[1 函数function 1](#_Toc514435396)

[1 什么是函数 1](#_Toc514435397)

[2.函数的调用 1](#_Toc514435398)

[May14 5](#_Toc514435399)

[1.return语句 5](#_Toc514435400)

[2 python函数的参数传递 6](#_Toc514435401)

[2.1 传递方式 6](#_Toc514435402)

[**2.1.1位置传参** 7](#_Toc514435403)

[**2.1.2序列传参** 7](#_Toc514435404)

[**2.1.3关键字传参** 7](#_Toc514435405)

[**2.1.4字典关键字传参** 8](#_Toc514435406)

[**2.1.5函数的综合传参** 8](#_Toc514435407)

[3函数的缺省函数 9](#_Toc514435408)

[4函数的形参定义方式 10](#_Toc514435409)

[**4.1星号元组形参** 10](#_Toc514435410)

[**4.2命名关键字形参** 11](#_Toc514435411)

[**4.3双星号形参** 11](#_Toc514435412)

[5函数的参数说明 12](#_Toc514435413)

[6函数的不定长参数 12](#_Toc514435414)

[May15 15](#_Toc514435415)

[1 全局变量和局部变量 15](#_Toc514435416)

[1.1 局部变量 15](#_Toc514435417)

[1.2全局变量 15](#_Toc514435418)

[2 globals和locals函数 15](#_Toc514435419)

[3 函数变量 16](#_Toc514435420)

[4 函数的嵌套定义 19](#_Toc514435421)

[5 python 作用域 19](#_Toc514435422)

[6 global语句 21](#_Toc514435423)

[7 nonlocal语句 21](#_Toc514435424)

[8 lambda表达式（又称匿名函数） 23](#_Toc514435425)

[9 eval()和exec()函数 24](#_Toc514435426)

[May16 27](#_Toc514435427)

[1 函数式编程 27](#_Toc514435428)

[1.1 高阶函数High order function 27](#_Toc514435429)

[**1.1.1 map函数** 27](#_Toc514435430)

[**1.1.2 filter函数** 29](#_Toc514435431)

[**1.1.3 sorted函数** 30](#_Toc514435432)

[2 递归函数 recursion 31](#_Toc514435433)

[3 闭包 closure 32](#_Toc514435434)

[May17 34](#_Toc514435435)

[1 装饰器decorators（专业提高篇） 34](#_Toc514435436)

[2 函数的文档字符串 35](#_Toc514435437)

[2.1函数的\_\_doc\_\_属性 36](#_Toc514435438)

[2.2 函数的\_\_name\_\_属性： 36](#_Toc514435439)

[3函数定义语句（def语句的语法） 36](#_Toc514435440)

[4 面试题 36](#_Toc514435441)

[5 模块Module 37](#_Toc514435442)

[5.1 模块的导入语句 37](#_Toc514435443)

[**5.1.1 dir(obj)** 37](#_Toc514435444)

[6 数学模块 39](#_Toc514435445)

[数学模块用法： 39](#_Toc514435446)

[7时间模块time 40](#_Toc514435447)

[May18 43](#_Toc514435448)

[1 系统模块 sys 43](#_Toc514435449)

[1.1sys模块的变量 43](#_Toc514435450)

[1.2 sys模块的方法 43](#_Toc514435451)

[2 自定义模块 44](#_Toc514435452)

[**2.1 模块化编程的优点：** 45](#_Toc514435453)

[**2.2 import语句搜索模块的路径顺序** 45](#_Toc514435454)

[3 模块的加载过程 45](#_Toc514435455)

[**3.1模块的重新加载** 46](#_Toc514435456)

[4 模块被导入和执行的过程 46](#_Toc514435457)

[5 模块的属性 46](#_Toc514435458)

[5.1 \_\_doc\_\_属性 46](#_Toc514435459)

[5.2 \_\_file\_\_属性 46](#_Toc514435460)

[5.3 \_\_name\_\_属性 46](#_Toc514435461)

[6 模块的\_\_all\_\_列表 47](#_Toc514435462)

[7 模块的隐藏属性 47](#_Toc514435463)

[8 随机模块random 47](#_Toc514435464)

[9 包（模块包）package 48](#_Toc514435465)

[9.1 包的导入 49](#_Toc514435466)

1 函数function

1 什么是函数

函数是可以重复执行的语句块，可以重复使用

作用：

1.用于封装语句块，提高代码的重用性

2.定义用户级别的函数

函数定义（创建）语句 def 语句的语法：

def 函数名（字母\_数字）（形参列表）：

语句块

说明：

1.函数的名字就是语句块的名称

2.函数名的命名规则与变量名相同（函数名必须为表示符）

3.函数有自己的名字空间，在函数外部不可以访问函数内部的变量，在函数内部可以访问函数外部的变量，通常让函数处理外部数据需要用参数给函数传入一些数据

4.函数的参数列表可以为空

5.语句部分不能为空，如果为空需要填充pass语句

示例：

#定义函数

def say\_hello():

print("hello world!")

print("hello tarena!")

print("hello everyone!!!")

#调用函数

say\_hello()

2.函数的调用

函数名（实际调用传递参数列表）

注：实际调用参数以后为实参

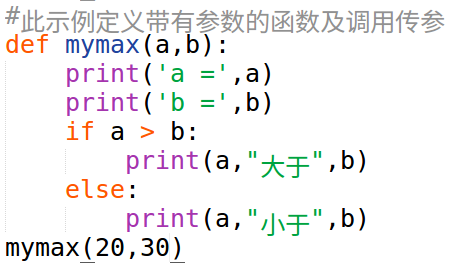
说明：

函数调用是一个表达式

如果没有return语句，此函数执行完毕后返回None对象

如果函数需要返回其他的对象需要用到return语句

示例：



练习：

1.写一个函数myfun，此函数用显示两个参数的相关信息。

函数：

def myfun(a,b):

此处自己实现

此函数给定两个参数，打印关于两个参数的信息：

1）打印两个参数的最大值

2）打印两个参数的和

3）打印两个参数的积

4）打印从a开始到b结束的所有偶数

如：

myfun（3,10）

打印如下：

最大值是10

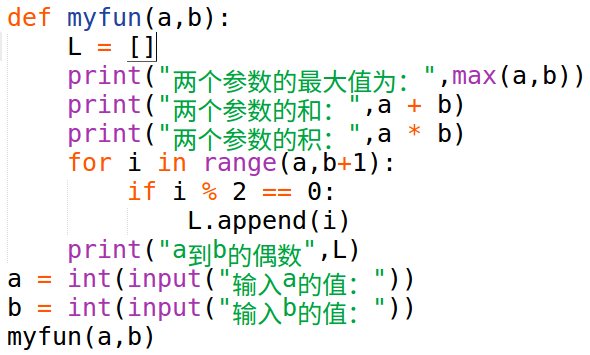
和是：13

积是：30

3到10之间的偶数是：4 6 8

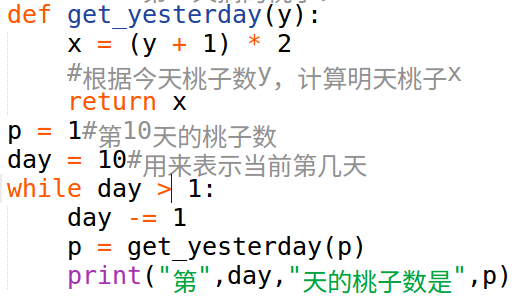
myfun（10,20）

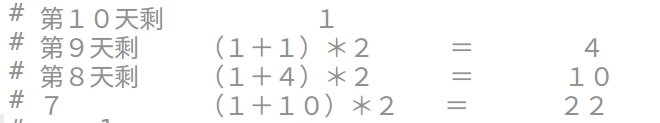
打印……



2.猴子吃桃，有一只小猴子，摘了很多桃。第一天吃了全部的一半，感觉不饱又吃了一个。第二天吃了剩下的一半，感觉不饱又吃了一个……以此类推。

到第十天，发现只剩了一个。请问第一天摘了多少桃。





3.完全数：

1+2+3=6(6为完全数)1,2,3都为6的因数（能被一个数x整除的数为y，则y为x的因数）

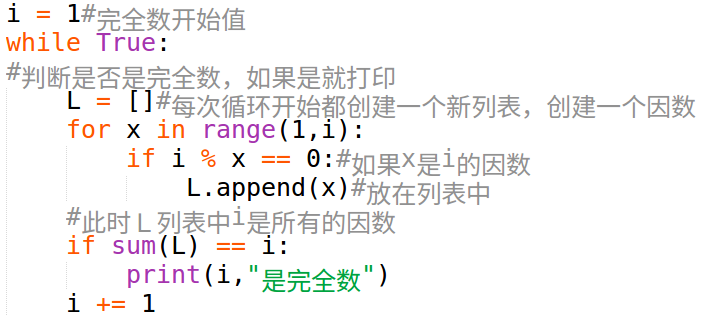
1 x 6 = 6

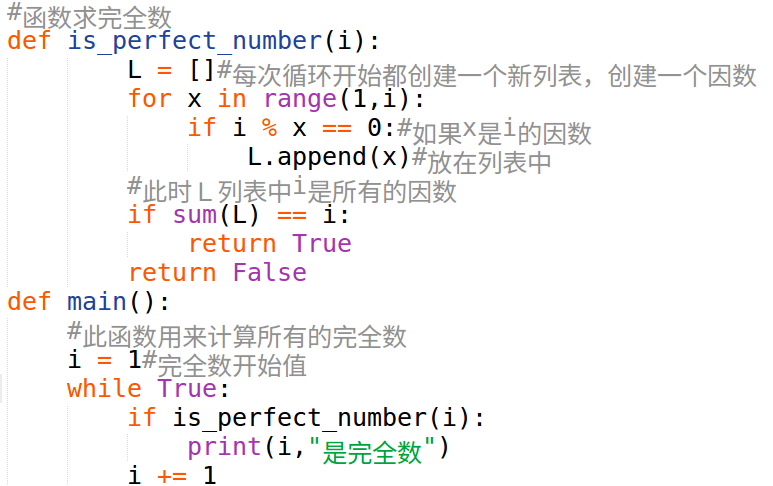
2 x 3 = 6

完全数是指除自身以外，所有因数相加之和等于自身的数。

求4~5个完全数并打印

答案：6 28 496





May14

1.return语句

语法：

return[表达式]

[ ]代表可以省略

作用：

用于函数中，结束当前函数的执行，返回到调用该函数的地方，同时返回一个对象的引用关系

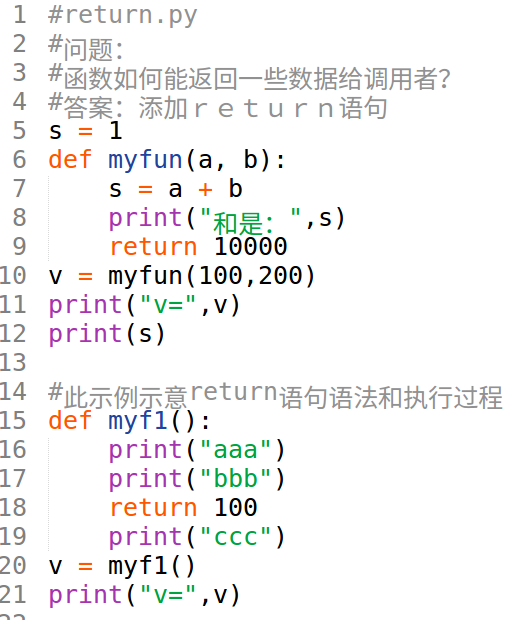
说明：

1.return语句后跟的表达式可以省略，省略后相当于return None

2.如果函数没有return语句，则函数执行完最后一条语句后返回None（相当于在最后加了一条return None语句）

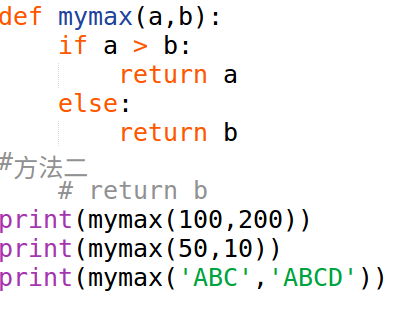
3.函数调用一定会返回一个对象的引用

示例：



练习：return的练习

写一个函数mymax，实现返回两个数的最大值：



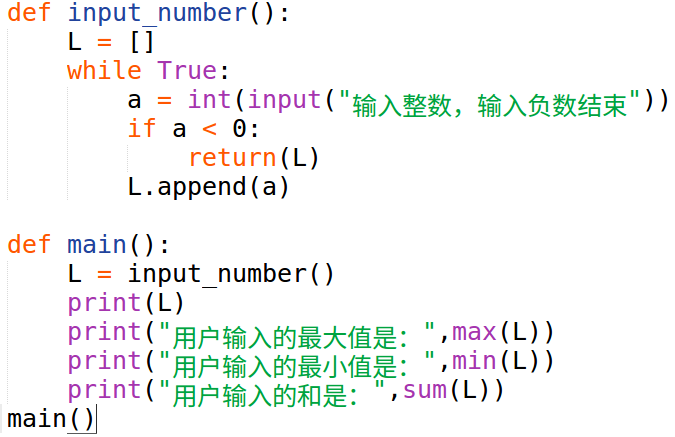
2.写一个函数input\_number

def input\_number():

……

此函数用来获取用户循环输入的整数，当用户输入负数时结束输入。

将用户输入数以列表的形式返回，再用内建函数max，min，sum展示出用户输入的最大、最小及和的值



2 python函数的参数传递

2.1 传递方式

位置传递

序列传递

关键字传参

字典关键字传参

**2.1.1位置传参**

实际参数（实参）的对应关系与形式参数（形参）的对应关系是按位置来依次对应的

示意：

def myfun(a,b,c):

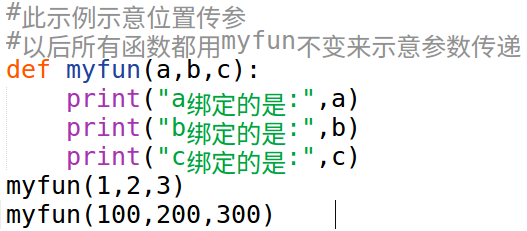
pass

myfun(1,2,3)

说明：

实际参数和形式参数通过位置进行传递匹配

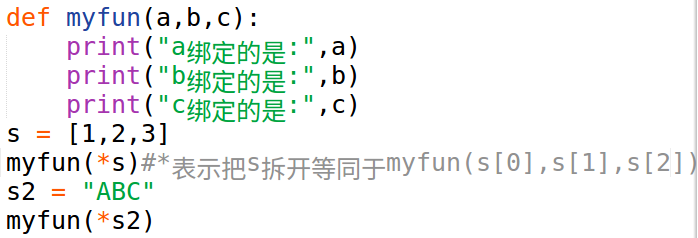
实参个数必须与形参个数相同



**2.1.2序列传参**

序列传参是指在函数调用过程中，用\*将序列拆解后按位置传参的方式进行参数传递

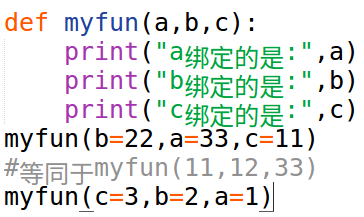
示例：



**2.1.3关键字传参**

关键字传参是指传参时，按着形参的名称给形参赋值。 实参和形参按名称进行匹配。

示例：



说明：实参和形参可以不按位置匹配

**2.1.4字典关键字传参**

是指实参为字典，将字典用\*\*拆解后进行关键字传参

示例：



说明：

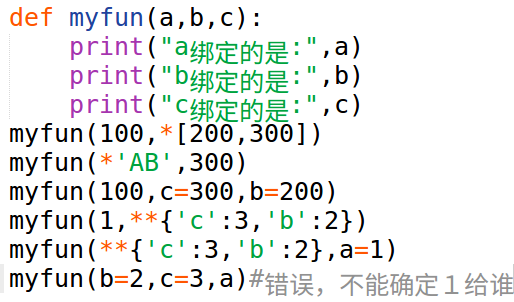
1.字典的键名和形参名必须一致

2.字典的键名必须为字符串

3.字典的键名要在形参中存在

**2.1.5函数的综合传参**

函数传参方式，在能确定形参能唯一匹配到相应实参的情况下可以任意组合

示例：

说明：先位置传参，后关键字传参

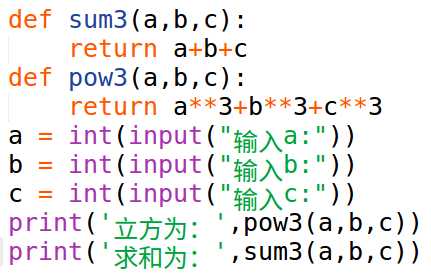
练习：

1.写一个函数sum3(a,b,c): 用于返回三个数的和

写一个函数pow3(x)用于返回x的三次方

1）用以上函数计算1\*\*3+2\*\*3+3\*\*3

2）计算1 + 2 +3的和的立方



------------------------------------------函数的形参（如何接收实参）------------------------------------

3函数的缺省函数

语法：

def 函数名 （形参名1=默认实参1，

形参名2=默认实参2，……）：

示例：

def info(name,age1=1,address=”未填写”)：

print（name,”今年”,age,”岁,家庭地址是：”,address）

info(‘tarena’,15)

info(‘小魏’,20,’北京朝阳区’)

info(‘小李’)

说明：

1.缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则基右侧的所有参数都必须有缺省参数

如：def test(a,b=10,c):#错误的

pass

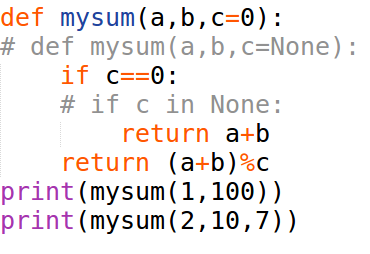
2.缺省参数可以有0个或者多个，甚至全部都有缺省参数

练习：

1.写一个函数，mysum(),可以传入两个实参或者三个实参：

1）如果传入两个实参，则返回两个实参的和

2）如果入三个实参，则返回前两个实参的和后对第三个实参求余的结果



4函数的形参定义方式

位置形参

星号元组形参

命名关键字形参

双星号字典形参

位置形参：

语法：

def 函数名（形参名1，形参名2……）：

语句块

**4.1星号元组形参**

语法

def 函数名（\*元组形参名）：

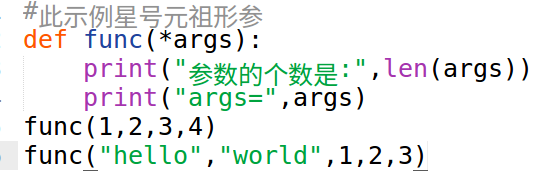
作用：

收集多合的位置传参

说明：

元组形参名通常用: ’args’

示例：



练习：

写一个函数，mysum，可以传入任意个实参的数字，返回所有实参的和



**4.2命名关键字形参**

语法：

def 函数名(\*,命名关键字形参):

语句

或

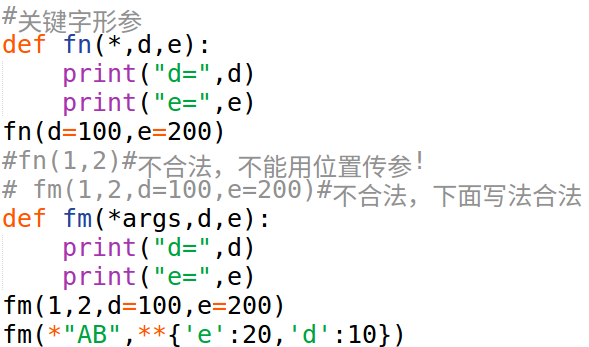
def 函数名(\*args,命名关键字形参):

语句

作用：

所有的参数都必须用关键字传参或字典关键字传参传递

示例：



**4.3双星号形参**

语法：

def 函数名(\*\*字典形参名)：

语句

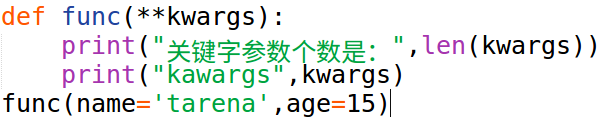
作用：

收集多合的关键字传参

说明：

通常字典形参名：kwargs

示例：





5函数的参数说明

缺省参数，位置形参，星号元组形参，命名关键字形参和双星号字典形参可以混合使用

函数参数自左至右的顺序为：

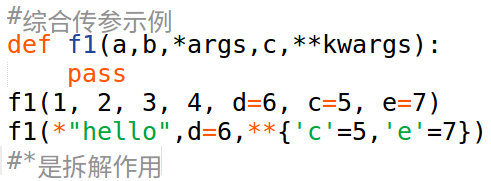
位置形参

星号元组形参

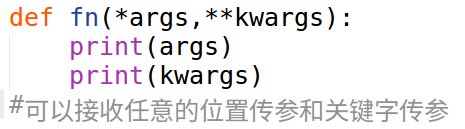
命名关键字形参

双星号字典形参

综合示例：



6函数的不定长参数



练习：

写一个myrange函数，此函数返回一个符合range规则的整数列表

如：

L = myrange(3)

print(L) #[0,1,2]

L = myrange(3,6)

print(L) #[3,4,5]

L = myrange(1,10,3)

print(L) #[1,4,7]



练习：

1.素数prime函数练习

1)写一个函数isprime（x），判断x是否为素数，如果是素数返回True，否则返回False

2)写一个函数prime\_m2n(m,n)，返回从m开始到n结束（不包含n）的范围内的素数列表

如：L = prime\_m2n(5,10)

print(L) #[2,3,5,7]

3)写一个函数primes(n)，返回指定范围内素数（不包含n）的全部素数的列表，并打印这些素数

如：L = prime（20）

print（L） #[2,3,5,7,11,13,17,19]

1.打印100以内的全部素数

2.打印100以内全部素数和

2.修改之前的学生信息管理程序

编写两个函数用于封装 录入学生信息的功能

1) input\_student()

#此函数获取学生信息，并返回学生信息的字典列表

2)def output\_student(L):

#以表格形式打印学生信息

……

验证测试：

L = input\_student()

output\_student(L)

print(“再添加几个学生信息”)

L += input\_student()

print(“添加学生后的信息如下：”)

out\_student(L)

May15

1 全局变量和局部变量

1.1 局部变量

定义在函数内部的变量称为局部变量（函数的形参也是局部变量）

局部变量只能在函数内部使用

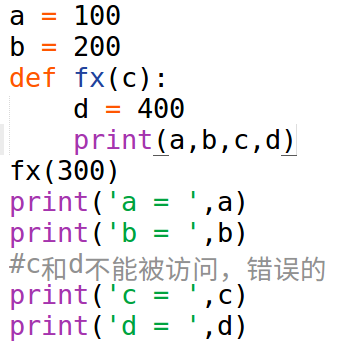
局部变量在函数调用时才能够被创建，在函数调用之后会自动销毁

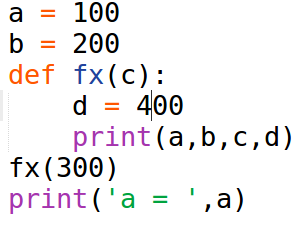
1.2全局变量

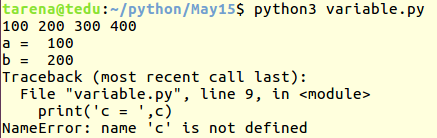
定义在函数外部，模块内部的变量称为全局变量

全局变量，所有的函数都可以直接访问（但函数内部不能将其直接赋值）

示例：







局部变量说明：

1.在函数内首次对变量赋值是创建局部变量，再次为变量赋值是修改局部变量的绑定关系

2.在函数内部的赋值语句不会对全局变量造成影响

3.局部变量只能在其被声明的函数内部访问，而全局变量可以在整个模块范围内访问

2 globals和locals函数

globals（）返回当前全局作用域内变量的字典

locals （）返回当前局部作用域内量的字典

示例：

a = 1

b = 2

c = 3

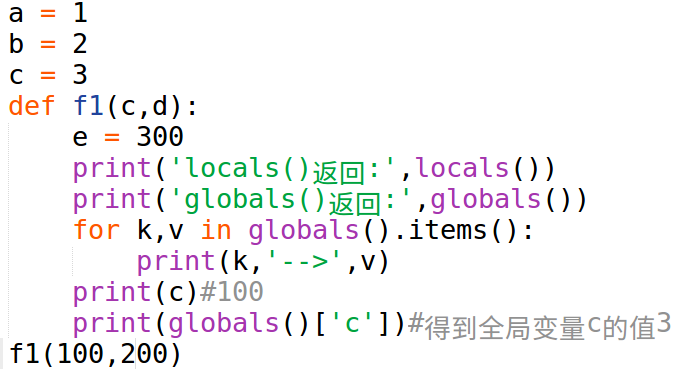
def f1(c,d):

e = 300

print('locals()返回:',locals())

print('globals()返回:',globals())

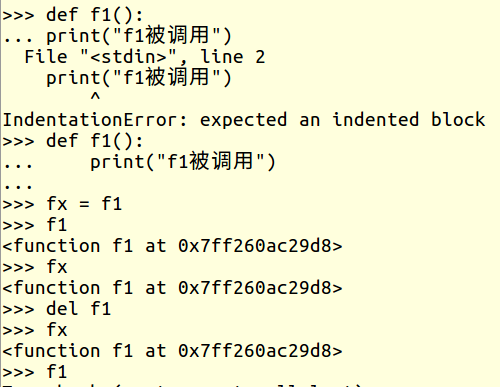
f1(100,200)



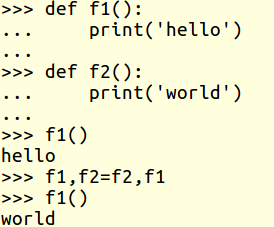
3 函数变量

函数名是变量，它在创建函数时绑定一个函数

示例：

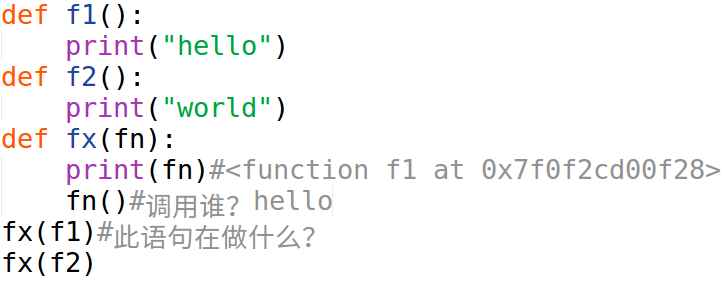


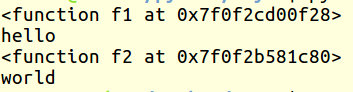
示例2：



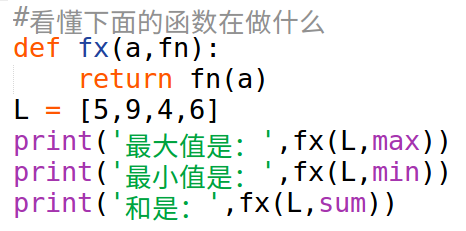
一个函数可以作为另一个函数的参数传递

示例：



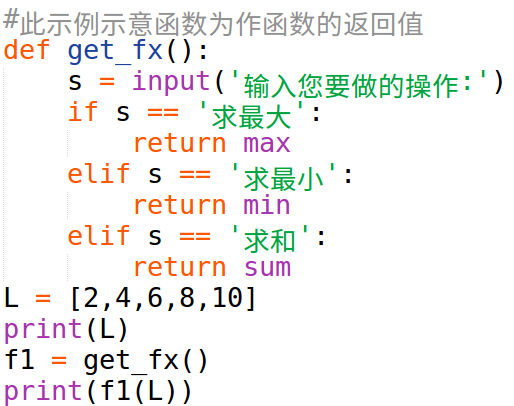


示例：



函数可以返回另一个函数（即：另一个函数可以返回一个函数）

示例：



练习：

写一个计算器解释执行器，已知有如下函数：

def myadd(x,y):

return x+y

def mymul(x,y):

return x\*y

def get\_op(s):#s代表操作字符串：‘加’，‘乘’

主函数:

def main():

while True:

s= input(‘请输入计算公式：’)#如:10加20

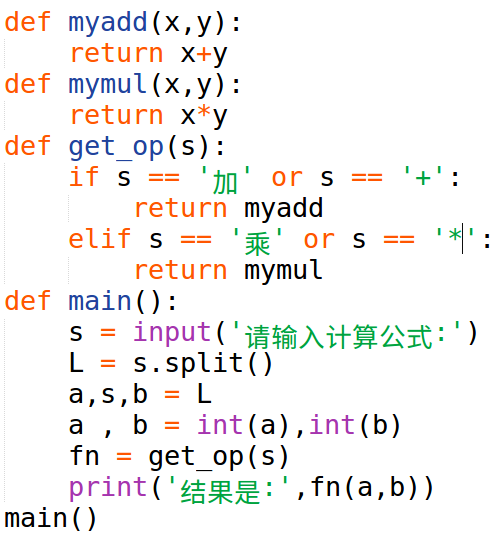
L =s.split()

a,s,b = L

a,b = int(a),int(b)

fn = get\_op(s)

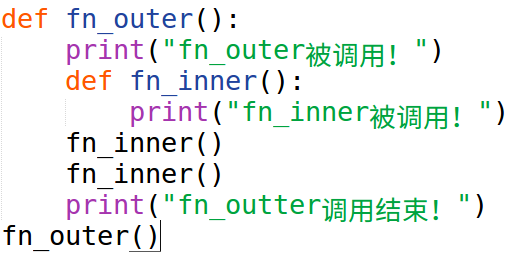
print(‘结果是：’,fn(a,b))#30

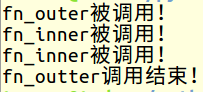


4 函数的嵌套定义

函数的嵌套定义是指一个函数里用def语句来创建其它的函数

示例：





5 python 作用域

作用域也叫名字空间，是访问变量时，查找变量名的范围空间

Python中的四个作用域LEGB

局部作用域 Local function

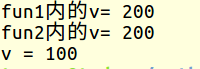
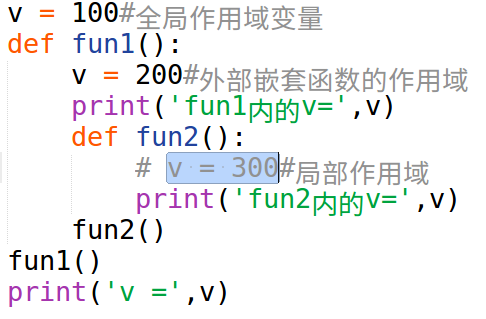
外部作用域 Enclosing Function Locaals

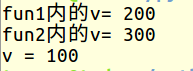
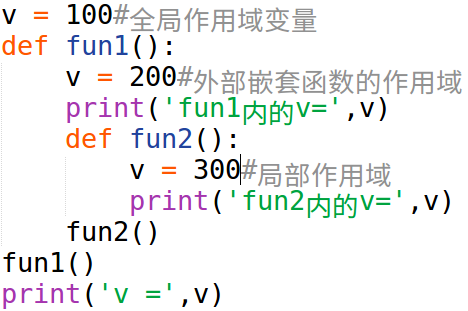
函数定义所在的模块（文件）的作用域 Global(Mudule)

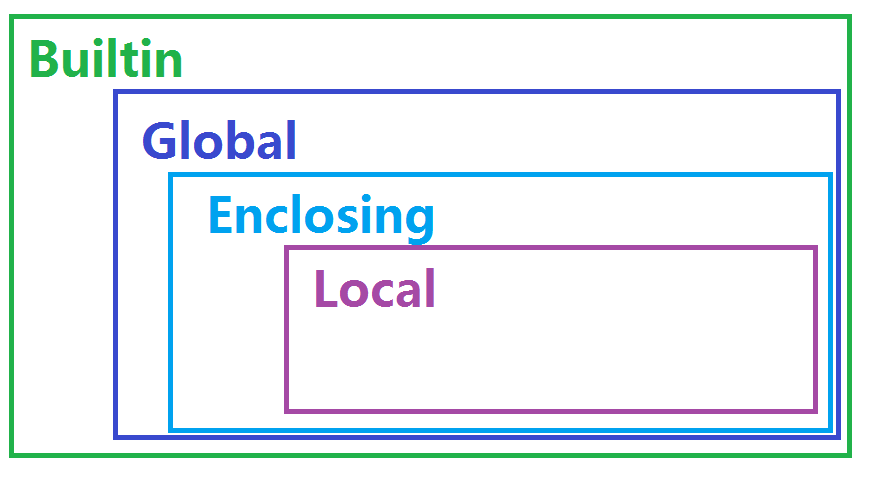
python内置模块作用域 Builtin(python)

以上作用域是从下到上的包含

访问是从内到外的

变量名的查找规则

L ------>E ------>G ------>B

在默认情况下，对变量名赋值会创建或改变本作用域内的变量

问题：

v = 100

def f():

v = 200

f()

print(v) #希望得到200，该怎么办？（目前为100）

6 global语句

作用：

1.告诉解释器，global语句声明一个或多个变量，这些变量的作用域为模块级作用域，也称作全局变量

2.全局声明（global）将赋值变量映身到模块问内部的作用域

语法：

global 变量1，变量2……

示例：

v = 100

def fn():

global v

v = 200

fn()

print(v) #200 对比上一个示例

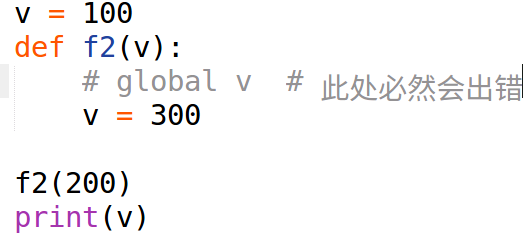
global说明：

1.全局变量如果要在函数内部被赋值，则必须经过全局声明（否则会被认为是局部变量）

2.全局变量在函数内部不经过声明就可以直接访问

3.不能先声明局部的变量，再用global声明为全局变量，此做法不符合规则

4.global变量列表里的变量不能出现在此作用域内的形参列表里



理由看说明4，由于形参在函数f2被定义为v所以再定义global V就会出错

7 nonlocal语句

作用：

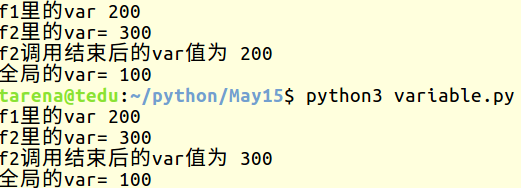
告诉解释器，nonlocal声明的变量不是局部变量，也不是全局变量，而是外部嵌套函数内的变量

语法：

nonlocal变量名1，变量名2……

示例：





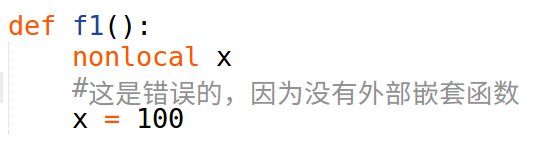
nonlocal说明：

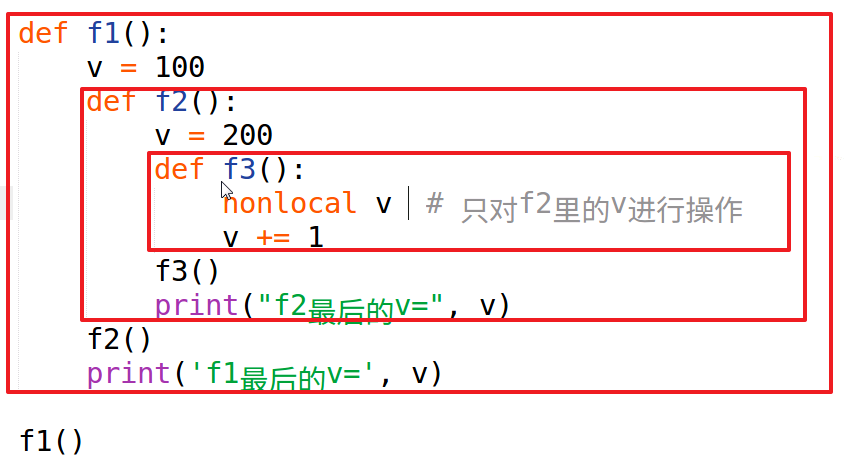
1.nonlocal语句只能在被嵌套函数内部进行使用

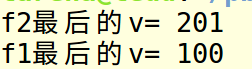
2.访问nonlocal变量将对外部嵌套函数的作用域的变量进行操作

3.当有两层或两层以上的函数嵌套时，访问nonlocal变量值对最近一层变量进行操作

4.nonlocal语句的变量列表里的变量名，不能出现在此函数的参数列表中









8 lambda表达式（又称匿名函数）

作用：

创建一个匿名函数对象

同def类似，但不提供函数名

语法：

lambda[参数1，参数2，……]:表达式(只能写一段)

[ ]里的内容可以省略

示例：

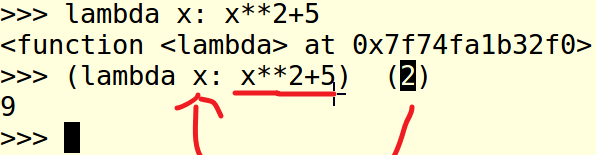
def myadd(x,y):

return x+y

#以上函数可以写为：

myadd = lambda x,y : x+y

print(‘2+3 = ’,myadd(x ,y))



语法说明：

1.lambda只是一个表达式，他用来创建一个函数对象

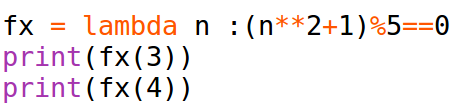
2.当lambda表达式调用时，先执行冒号后（：）的表达式，并返回表达式的结果的引用

3.lambda表达式创建的函数只能包含一条“表达式”

4.lambda比函数简单，且可以随时创建和销毁，有利于减少程序的偶合度

练习：

1.写一个lambda表达式，判断这个数的2次方+1是否能被5整除，如果能整除返回True，否则返回False



fx = lambda n:……

print(fx(3))#True

print(fx(4))#False

2.写一个lambda表达式，求两个变量的最大值：

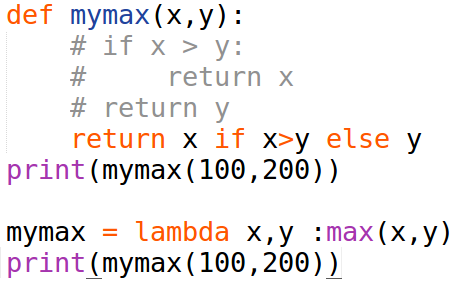
def mymax(x,y):

……

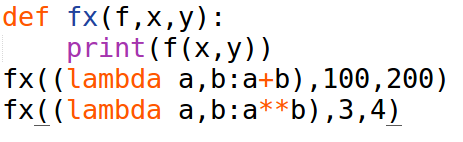
#或者用lambda

mymax = lambda……

print(mymax(100,200)) #200



看懂下面的程序在做什么？



语法说明第4条

9 eval()和exec()函数

eval()函数

格式：

eval(source,global=None,locals=None)

作用：

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。如果不是表达式就会出错。

示例：

x = 100

y = 200

a = eval(‘x+y’)

print(a)

exec()函数

作用：

把一个字符串当成程序来执行

格式：

exec(source,gloabls=None,locals=None)

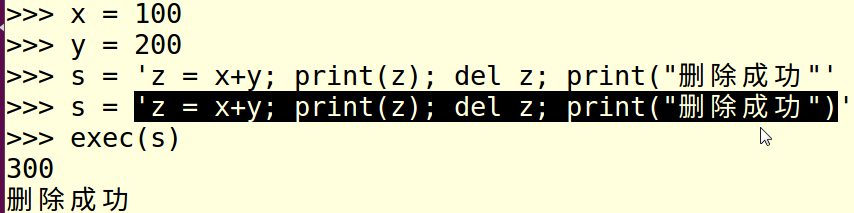
示例：

x = 100

y = 200

s = ‘z = x + y;print(z);del z; print(“删除成功”) ’

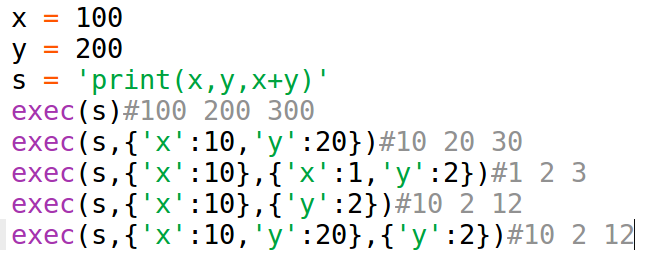
exec(s)#执行s绑定的语句



eval和exec的两个参数globals和locals

这两个参数是用来设置‘表达式’或‘程序’运行的全局变量和局部变量

示例：



有局部变量先访问局部变量，其次是全局变量

练习：

1.写一个函数mysum，要求给出一个数n，求

1 + 2 + 3 + 4 + ……+n的和

如：

print(mysum(100)) #5050

print(mysum(10)) #55

2.写一个函数myfac来计算n!(n的阶乘)

n! = 1\*2\*3\*4\*……\*n

如：

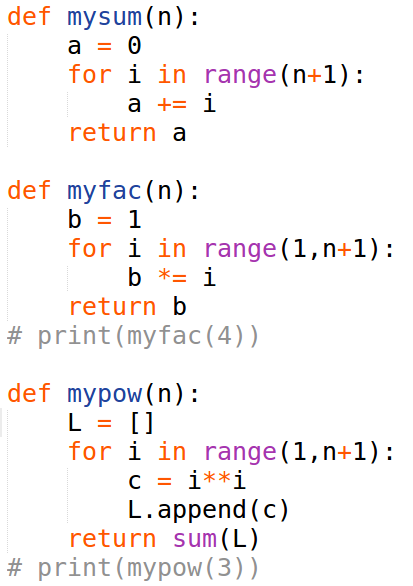
print(myfac(5)) #120

print(myfac(4)) #24

3.写一个函数,求

1 + 2 \*\* 2 + 3\*\*3 + …… + n\*\*n的和

（n能给个小点的数）



4.修改之前的学生信息管理程序，实现添加菜单和选择菜单操作功能：

菜单：

+------------------------------------------------------+

| 1）添加学生信息

| 2）查看所有学生信息

| 3）修改学生成绩

| 4）删除学生信息

| q）退出

+-------------------------------------------------------------+

请选择：1

请输入姓名：……

请选择：3

请输入修改学生的姓名：……

（要求每个功能都对应一个函数）

May16

1 函数式编程

函数式编程是指用一系列函数决定问题

1.函数是一等公民

2.函数本身可以赋值给变量，赋值后变量绑定函数

3.允许将函数本身作为参数传入另一个函数

4.允许返回一个函数

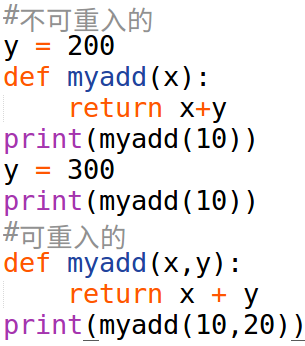
函数的可重入性：

可重入是指一个函数传的参数一定，则结果必须一定

要求：

def定义的函数不要访问除局部变量以外的变量

示例：



1.1 高阶函数High order function

什么是高阶函数：

满足下列一个条件的函数即为高阶函数：

1.函数接受一个或多个函数做为参数传入

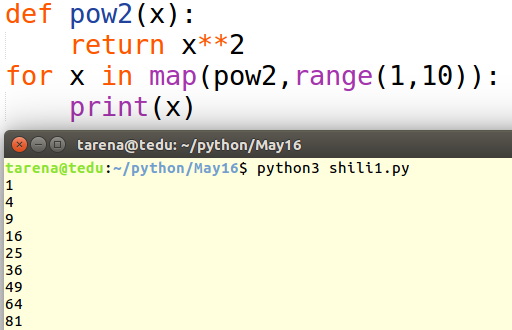
2.函数返回一个函数

python中内建的高阶函数 map filter sorted

**1.1.1 map函数**

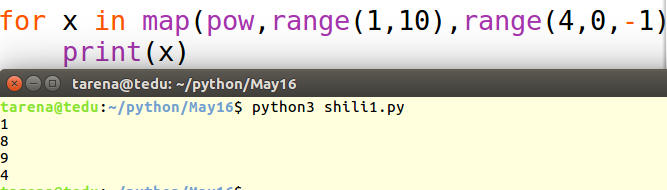
map(func,\*iterables)用函数和对可迭代对象中的每一个元素作为参数计算出新的可迭代对象，当最短的一个可迭代对象不再提供数据时，此可迭代对象生成结束

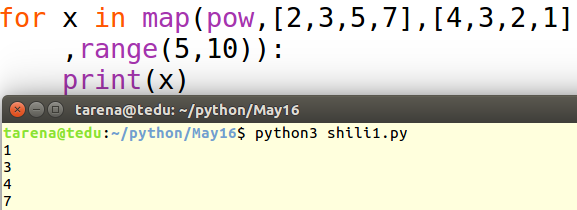
示例：



内建的pow(x,y,z=None) z是求余

#生成 1\*\*4 , 2\*\*3 , 3\*\*2 , 4\*\*1



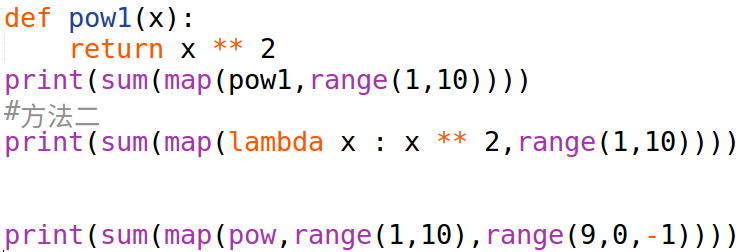


练习：

1.求1\*\*2+2\*\*2+3\*\*2+……+9\*\*2的和

2.求1\*\*3+2\*\*3+3\*\*3+……+9\*\*3的和

3.求1\*\*9+2\*\*8+3\*\*7+……+9\*\*1的和



**1.1.2 filter函数**

格式：

filter（func，iterable）

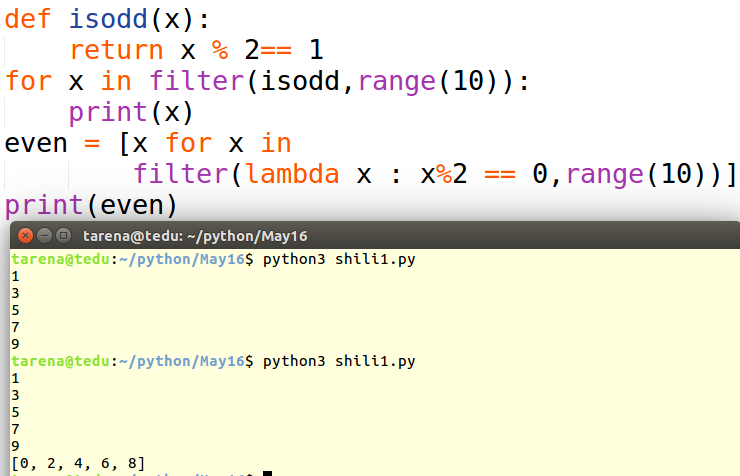
作用：

筛选可迭代对象iterable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

说明：

函数func将对每个元素进行求值，返回False则将此数据丢弃，返回True则保留此数据

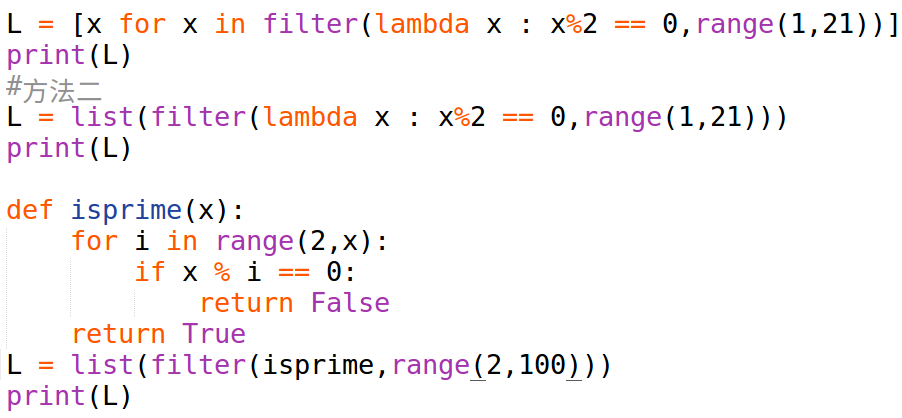
示例：



练习：

1.将1~20内的偶数用filter筛选出来，形成列表

2.用filter函数将1~100之间的所有素数放入到列表中



**1.1.3 sorted函数**

作用：

将原可迭代对象的数据进行排列，生成排序后的列表（不是可迭代对象）

格式：

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

说明：

iterable可迭代对象

key函数是用来提供一个参考值，这个参考值将作为排序的依据

reverse标志用来设置是否降序排序

示例：

L = [5,-2,-4,0,3,1]

L2 = sorted(L) 升序

L2 = sorted(L,reverse = True)降序

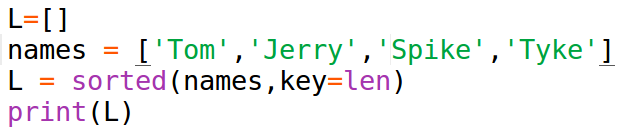
L2 = sorted(L,key=abs) #[0,-1,-2,3,-4,-5]

思考：

names = [‘Tom’,’Jerry’,’Spike’,’Tyke’]

sorted(names)

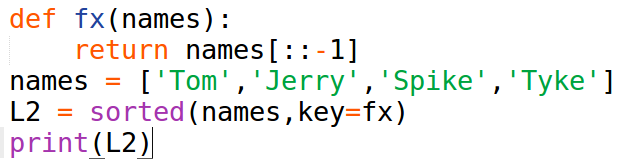
想要按名字的长度升序怎么办？



练习：

names = [‘Tom’,’Jerry’,’Spike’,’Tyke’]

让names排序，排序的依据是字符串的反序



练习：

写一个函数input\_student（）得到学生的姓名，成绩，年龄

L = input\_student() #输入一些学生信息

print(“按年龄从大到小排序”)

L2 = sorted(L,……)

output\_student(L2)

print(“按成绩从大到小排序”)

L3 = sorted(L,……)

output\_student(L3)

2 递归函数 recursion

函数直接或间接的调用自身

示例：

def fa():

fb()

def fb():

fa()

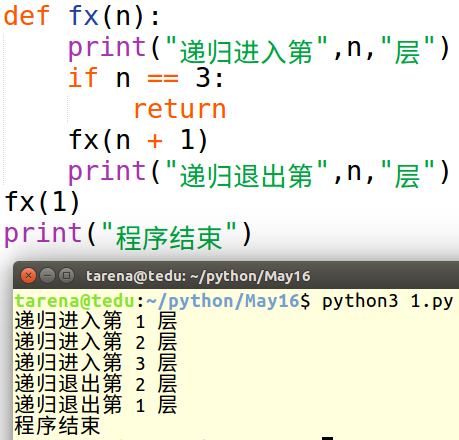
fa

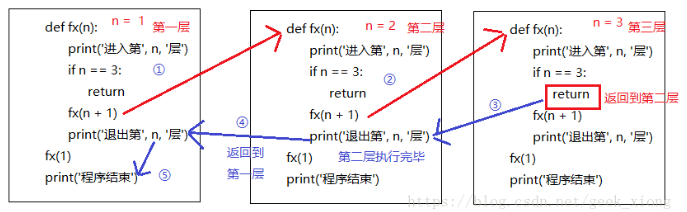
递归说明：

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

几乎所有的递归都能用while循环来代替

控制递归的层次的示例：





递归的优缺点

优点：

递归可以把问题简单化，让思路更为清晰，代码更简洁。

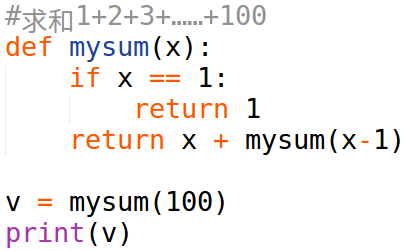
缺点：

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

递归函数的实现方法：

先假设函数已经实现

示例：



3 闭包 closure

将内嵌函数的语句和这些语句的执行环境打包在一起时，得到对象称为闭包（closure）

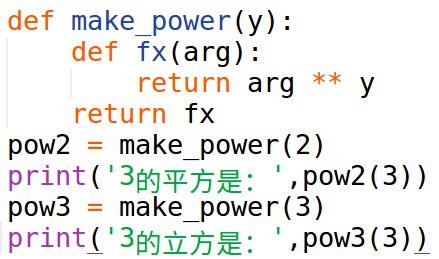
闭包必须满足三个条件：

1.必须有一个内嵌函数

2.内嵌函数必须引用外部函数中的变量

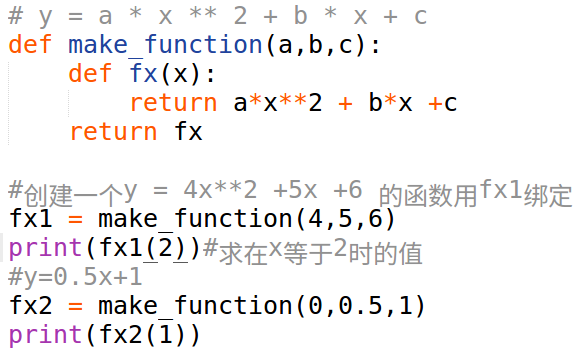
3.外部函数返回值必须是内嵌函数

示例：



示例：

用参数返回相应的数学函数的示例



练习：

1.编写函数求阶乘myfac（x）

2.写程序算出1~20的阶乘的和

1!+2!+3!+4!+……+20!

3.改写之前的学生理程序

1）成绩从高到低打印学生信息

2）成绩从低到高打印学生信息

3）年龄从高到低打印学生信息

4）年龄到低到高打印学生信息

要求原来的输入列表顺序不变

4.已知有列表：

L = [[3,5,8],10,[[13,14],15,18],20]

1）写一个函数print\_list(lst)打印出所有元素

print\_list(L)#打印……

2）写一个函数sum\_list(lst)

返回这个列表中所有元素的和

注：

type(x)可以返回一个变量的类型

如：

type(20) is int # 返回True

type([1,2,3]) is list # 返回True

May17

1 装饰器decorators（专业提高篇）

函数装饰器是指装饰的是一个函数，传入的是一个函数，返回的也是一个函数的函数

问题：

def say(x):

print(“ 您好:”,x)

say(‘老魏’)#您好 老魏

#希望打印的是：

开始问候：

您好：老魏

结束问候：

函数装饰器的语法：

def 装饰器函数名（参数）：

语句块

return 函数对象

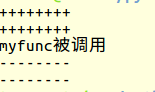
被装饰函数的语法

@装饰器函数名

def 函数名（形参列表）：

语句块

示例：



经过装饰后myfunc已经绑定了fx

装饰器 == 重新复制

银行业务：

存钱：

savemoney

取钱：

withraw



2 函数的文档字符串

函数内部，第一个没有赋值给任何变量的字符串为文档字符串

语法：

def 函数名（形参列表）:

’’’函数的文档字符串’’’

函数语句块

示例：

def cba():

‘这是一块文档字符串……’

pass

>>>help(cba)

2.1函数的\_\_doc\_\_属性

\_\_doc\_\_属性用于绑定该函数的文档字符串

示例：

def fx(a,b):

’’’这是函数的文档字符串

这是第二行…

’’’

print(fx.\_\_doc\_\_)

2.2 函数的\_\_name\_\_属性：

\_\_name\_\_属性用于绑定函数名的字符串

示例：

def fx():

pass

f1 = fx

print(f1.\_\_name\_\_)

3函数定义语句（def语句的语法）

[@装饰器名1]

[@装饰器名2]

…

def 函数名([位置形参],[\*[元组形参名]],[命名关键字形参]，[\*\*字典形参1]):

’’’文档字符串’’’

语句块

[ ]内的可以省略

4 面试题

L = [1，2，3]

def f(n,lst=[]):

lst.append(n)

print(lst)

f(4,L) 结果是什么？

f(5,L) 结果是什么？

f(100) 打印结果是什么？

f(200) 打印结果是什么？

说明：

默认参数（缺省参数）绑定在函数对象内部，随函数的声明一直存在

lst为缺省参数，就算不传参列表也存在

解决办法：

L = [1,2,3]

def f(n,lst=None):

if lst is None:

lst = []

lst.append(n)

print(lst)

f(4,L)

f(5,L)

f(100)

f(200)

5 模块Module

模块是一个包含有一系列数据，函数，类等组成的程序组

模块是一个文件，模块文件通常以.py结尾

作用:

让一些相关的数据，函数，类等有逻辑的组织在一起，是逻辑结构更加清晰

模块中的数据，函数和类等可提供给其他模块或程序使用

模块的分类：

1.内置模块（builtins），在解析器的内部可以直接使用

2.标准库模块，安装python时已安装且可直接使用

3.第三方模块（通常为开源），需要自己安装

4.用户自己完成的模块（可以作为其他人的第三方模块）

5.1 模块的导入语句

import语句

语法：

import 模块名 1[as 模块新名1][,模块名2 as 模块新名2],……

示例：

#导入教学模块

import math

#导入系统模块sys和os模块

import sys , os

作用：

降某模块整体导入到当前模块

用法：

模块·属性名

**5.1.1 dir(obj)**

函数返回模块所有属性的字符串列表

dir([对象])返回一个字符串列表

作用：

一、如果没有参数调用，则返回当前作用域内的所有变量的列表

二、如果给定一个对象作为参数，则返回这个对象的所有变量的列表

1.对于一个模块，返回这个模块的全部属性

2.对于一个类对象，返回类对象的所有变量，亲递归基类对象的所有变量

3.对于其他对象，返回所有的变量，类变量，基类变量

help(obj) 可以查看模块相关的文档字符

练习：

1.输入一个正方形的周长，输出正方形的面积

2.输入一个圆的半径，打印出这个圆的面积

3.输入一个正方形的面积，打印出这个正方形的周长

（要求用math模块内的函数和变量）

import math

length = float(input('输入正方形的周长：'))

area = math.pow(length/4,2)

print('正方形的面积：',area)

from inport语句

语法：

from模块名 import模块属性名1[as 属性新名1]

[,模块属性名2[as 属性新名2],……]

作用：

将模块内的一个或多个属性导入到当前模块的作用域

示例：

from math import pi

from math import pow ,sqrt

from math import factorial as fac

from import \*语句

语法：

from 模块名 import\*

作用：

将某模块的所有属性都导入到当前模块

示例：

from math imoprt \*

6 数学模块

模块名：math

#### 数学模块用法：

import math

# 或

from math import \*

| **变量** | **描述** |
| --- | --- |
| math.e | 自然对数的底e |
| math.pi | 圆周率pi |

| **函数名** | **描述** |
| --- | --- |
| math.ceil(x) | 对x向上取整，比如x=1.2，返回2 |
| math.floor(x) | 对x向下取整，比如x=1.2，返回1 |
| math.sqrt(x) | 返回x的平方根 |
| math.factorial(x) | 求x的阶乘 |
| math.log(x[, base]) | 返回以base为底x的对数, 如果不给出base,则以自然对数e为底 |
| math.log10(x) | 求以10为底x的对数 |
| math.pow(x, y) | 返回 x\*\*y (x的y次方) |
| math.fabs(x) | 返回浮点数x的绝对值 |
| 角度和弧度degrees互换 |  |
| math.degree(x) | 将弧度x转换为角度 |
| math.radians(x) | 将角度x转换为弧度 |
| 三角函数 |  |
| math.sin(x) | 返回x的正弦(x为弧度) |
| math.cos(x) | 返回x的余弦(x为弧度) |
| math.tan(x) | 返回x的正切(x为弧度) |
| math.asin(x) | 返回x的反正弦(返回值为为弧度) |
| math.acos(x) | 返回x的反余弦(返回值为为弧度) |
| math.atan(x) | 返回x的反正切(返回值为为弧度) |

7时间模块time

此模块提供了时间相关的函数，且一直可用

* 此模块提供了时间相关的函数，且一直可用

**时间简介**

* 公元纪年是从公元 0000年1月1日0时开始的
* 计算机元年是从1970年1月1日0时开始的,此时时间为0,之后每过一秒时间+1
* UTC 时间 (Coordinated Universal Time) 是从Greenwich时间开始计算的.  
  UTC 时间不会因时区问题而产生错误
* DST 阳光节约时间(Daylight Saving Time)，又称夏令时, 是一个经过日照时间修正后的时间

**时间元组**

* 时间元组是一个9个整型元素组成的,这九个元素自前至后依次为:
  + 四位的年(如: 1993)
  + 月 (1-12)
  + 日 (1-31)
  + 时 (0-23)
  + 分 (0-59)
  + 秒 (0-59)
  + 星期几 (0-6, 周一是 0)
  + 元旦开始日 (1-366)
  + 夏令时修正时间 (-1, 0 or 1).
* 注：
  + 如果年份值小于100,则会自动转换为加上1900后的值

**模块名: time**

**时间模块用法：**

import time

# 或

from time import xxx

# 或

from time import \*

| **变量** | **描述** |
| --- | --- |
| time.altzone | 夏令时时间与UTC时间差(秒为单位) |
| time.daylight | 夏令时校正时间 |
| time.timezone | 本地区时间与UTC时间差(秒为单位) |
| time.tzname | 时区名字的元组， 第一个名字为未经夏令时修正的时区名, 第一个名字为经夏令时修正后的时区名 |

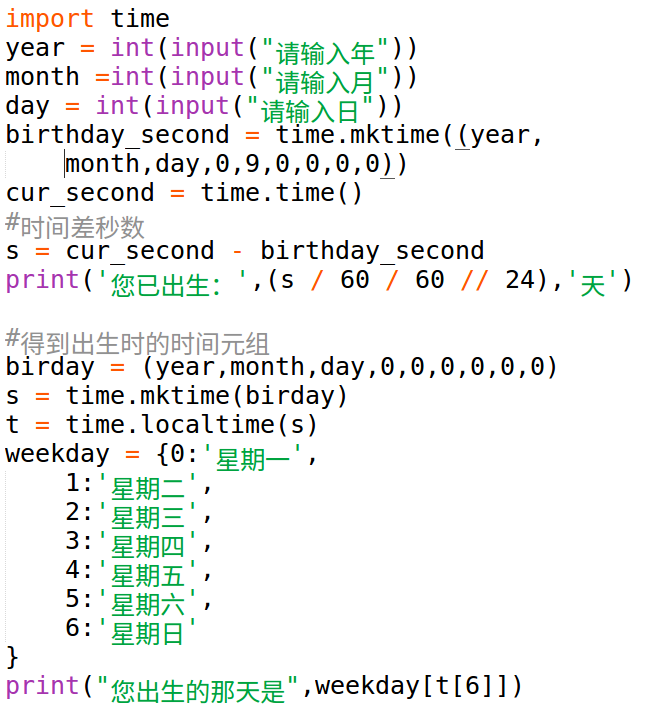
注： CST为中国标准时间(China Standard Time UTC+8:00)

| **函数名** | **描述** |
| --- | --- |
| time.time() | 返回从计算机元年至当前时间的秒数的浮点数(UTC时间为准) |
| time.sleep(secs) | 让程序按给定秒数的浮点数睡眠一段时间 |
| time.gmtime([secs]) | 用给定秒数转换为用UTC表达的时间元组 (缺省返回当前时间元组) |
| time.asctime([tuple]) | 将时间元组转换为日期时间字符串 |
| time.mktime(tuple) | 将本地日期时间元组转换为新纪元秒数时间(UTC为准) |
| time.localtime([secs]) | 将UTC秒数时间转换为日期元组（以本地时间为准) |

写一个程序，输入你的出生日期

1）算出你已经出生了多少天

2）算出你出生那天是星期几？



练习：

1.写一个程序，以电子时钟格式打印时间：

时间格式为：

HH:MM:SS 时间每隔一秒刷新一次

2.编写一个闹钟程序，启动时设定定时时间，到时候打印出一句话，然后程序退出

3.请编写函数fun，其功能是计算下列多项式的和 math.factorial

s = 1 + 1/1! + 2/2!+ + n/n!到100

May18

1 系统模块 sys

此模块全部是运行时系统相关信息

1.1sys模块的变量

| **变量** | **描述** |
| --- | --- |
| sys.path | 模块搜索路径 path[0] 是当前脚本程序的路径名，否则为 '' |
| sys.modules | 已加载模块的字典 |
| sys.version | 版本信息字符串 |
| sys.version\_info | 版本信息的命名元组 |
| sys.platform | 操作系统平台名称信息 |
| sys.argv | 命令行参数 argv[0] 代表当前脚本程序路径名 |
| sys.copyright | 获得Python版权相关的信息 |
| sys.builtin\_module\_names | 获得Python内建模块的名称（字符串元组） |
| 标准输入输出时会用到 | |
| sys.stdin | 标准输入文件对象，多用于input() |
| sys.stdout | 标准输出文件对象,多用于print() |
| sys.stderr | 标准错误输出文件对象, 用于输出错误信息 |

#### 1.2 sys模块的方法

| **函数名** | **描述** |
| --- | --- |
| sys.exit([arg]) | 退出程序，正常退出时sys.exit(0) |
| sys.getrecursionlimit() | 得到递归嵌套层次限制（栈的深度） |
| sys.setrecursionlimit(n) | 得到和修改递归嵌套层次限制（栈的深度） |

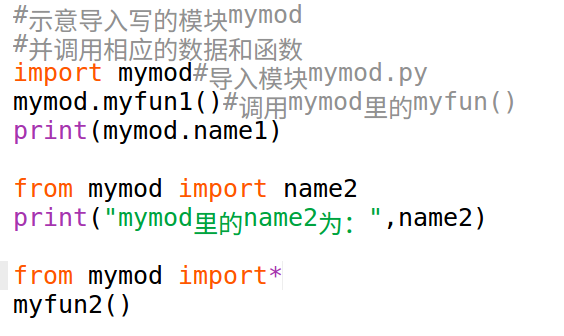
2 自定义模块

用来导入和测试自定义模块

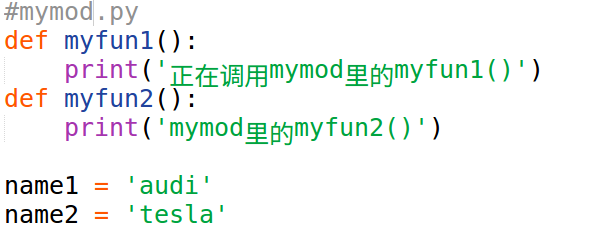
自定义模块的模块名必须符合“标识符”的命名规则（同变量名）

模块有各自独立的作用域

示例：



=======================================================================





**2.1 模块化编程的优点：**

1.有利于多人合作开发

2.使代码更易于维护

3.提高代码的复用率

4.有利于解决变量名冲突问题

**2.2 import语句搜索模块的路径顺序**

1.搜索程序运行时的路径（当前路径）

2.sys.path提供的路径

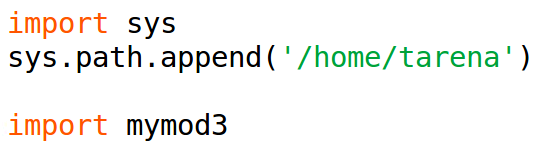
3.搜索内置模块

sys.path是一个存储模块搜索路径的列表，

1.可以把自定义的模块放在相应的路径下可以导入

2.可以把自己模块的路径添加在sys.path列表中

设置路径



将/home/tarena 路径加入模块路径

3 模块的加载过程

1.在模块导入时，模块所有语句会执行

2.如果一个模块已经导入，则再次导入时不会重新执行模块内的语句

**3.1模块的重新加载**

import mymod3

import imp

imp.reload(mymod3)#重新加载mymod3模块

4 模块被导入和执行的过程

1.先搜索相关路径找模块（.py）

2.判断是否有此模块对应的.pyc文件，如果存在pyc文件，且比.py文件新，直接加载.pyc文件

3.否则用.py文件生成.pyc后再进行加载

pyc模块的编译文件

编译 解释执行

mymod1.py-------->mymod1.pyc----------->python3

5 模块的属性

属性的实质是模块内的全局变量

模块内预置的属性

5.1 \_\_doc\_\_属性

作用：

用来绑定模块的文档字符串

模块内第一个没有复制给任何变量的字符创维模块的文档字符串

5.2 \_\_file\_\_属性

\_\_file\_\_绑定模块对应的文档路径名

1.对于内建模块，不绑定路径（不显示其路径）

2.对于其他模块，绑定路径名的字符串

5.3 \_\_name\_\_属性

此属性用来记录模块的自身名字

作用：

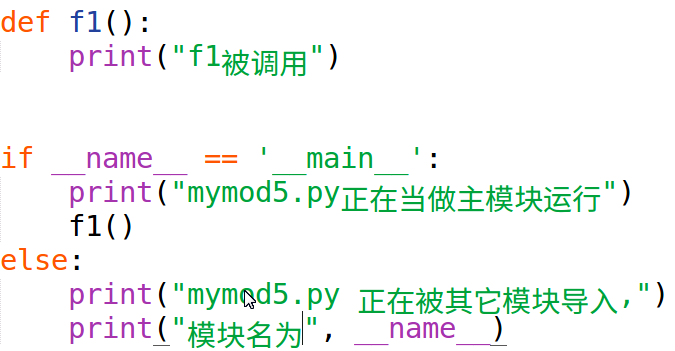
1.记录模块名

2.用来判断是否为主模块（最先运行的模块）

说明：

当此模块为主模块时，\_\_name\_\_绑定 ‘\_\_main\_\_’

当此模块不是主模块时，此属性绑定模块名



6 模块的\_\_all\_\_列表

模块中的\_\_all\_\_列表是一个用来存放可到处属性的字符串列表

作用：

当用from xxx import\*语句导入时，只导入\_\_all\_\_列表内的属性

示例：



7 模块的隐藏属性

模块中以‘\_’开头的属性，在from import\* 语句导入时，将不被导入，通常称这些属性为隐藏属性

示例：



8 随机模块random

作用：

用于模块或生成随机输出的模块

import random as R

| **函数名** | **描述** |
| --- | --- |
| R.random() | 返回一个[0, 1) 之间的随机实数 |
| R.uniform(a,b) | 返回[a,b) 区间内的随机实数 |
| R.randrange([start,] stop[, step]) | 返回range(start,stop,step)中的随机数 |
| R.choice(seq) | 从序列中返回随意元素 |
| R.shuffle(seq[, random]) | 随机指定序列的顺序(乱序序列） |
| R.sample(seq,n) | 从序列中选择n个随机且不重复的元素 |
| R.getrandbits(nbit) | 以长整型的形式返回用nbit位来表示的随机数 |
| R.seed(a=None) | 用给定的数a设置随机种子,不给参数a则用当前时间设置随机种子 |

练习：

猜数字游戏：

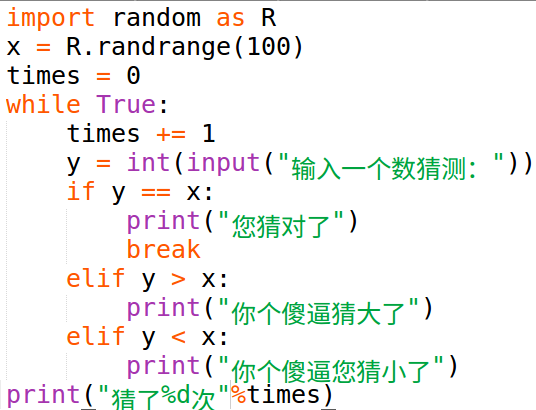
写程序，随机生成一个0~100之间的数用变量x绑定，循环让用户输入一个数用y绑定。输出猜数字的结果。

1.如果y等于生成的数x，则提示“您猜对了”，打印出猜测的次数并退出。

2.如果y小于x则提示“您猜小了”，然后继续猜

3.如果y大于x则提示，“您猜大了”，然后继续猜

猜对后退出程序并打印次数



原理：二分查找（计算机的查找方式，前提是排好序）

9 包（模块包）package

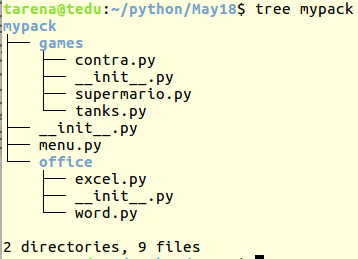
包是将模块以文件夹的组织形式进行分组管理的方法

作用：

将一系列模块进行分类管理，有利于防止命名冲突，可以在需要时家在一个或部分模块而不是全部模块。

示例：

mypack/



\_\_init\_\_.py

menu.py

games/

\_\_init\_\_.py

contra.py

supermario.py

tanks.py

office/

\_\_init\_\_.py

excel.py

word.py

文件夹内有 \_\_init\_\_.py 就可以看做包来用

\_\_init\_\_.py文件

常规包内必须存在的文件

\_\_init\_\_.py会在包加载时被自动调用

作用：

编写此包的内容

在内部填写文档字符串

在\_\_init\_\_.py内可以加载此包所以来的一些其他模块

9.1 包的导入

用三条import语句可以导入（同模块的导入规则）

import 包名 [as 包别名]

import 包名 .模块名[as 模块新名]

import 包名 .子包名 .模块名

from包名import 模块名[as 模块新名]

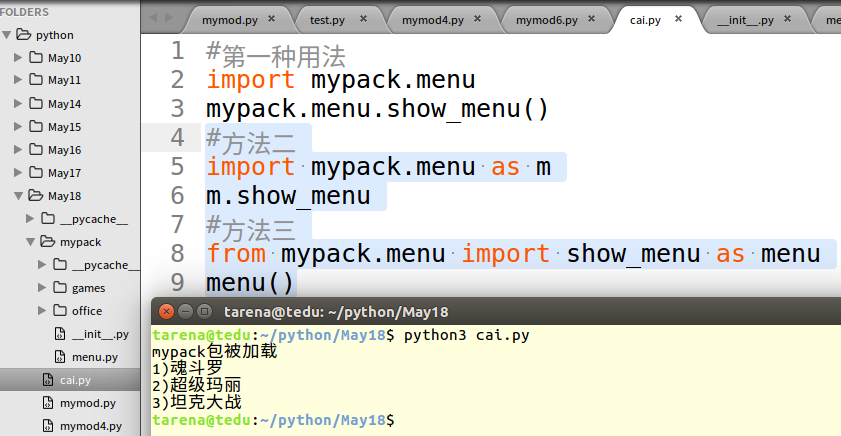
from包名 .子包名import模块名[as模块新名]

from 包名 .子包名 .模块名import属性名[as属性新名]

from 包名 import\*

from 包名.模块名import\*

……



练习：

1.把学生管理系统划分为为模块（把相关操作放在一个模块内）：

建议：

main.py放主事件循环

menu.py放show\_menu函数

student\_info.py 放学生信息相关操作

2.模拟斗地主发牌，牌共54张

黑桃(‘\u2660’),梅花（’\u2663’）,方块（‘\u2665’）,红桃(‘\u2666’)

大小王

A2-10JQK

三个人玩，每个人发17张牌，底牌留三张

操作：

输入回车：打印第一个人的17张牌

输入回车：打印第二个人的17张牌

输入回车：打印第三个人的17张牌

输入回车：打印三张底牌

May21

1 \_\_init\_\_.py 内的\_\_all\_\_列表

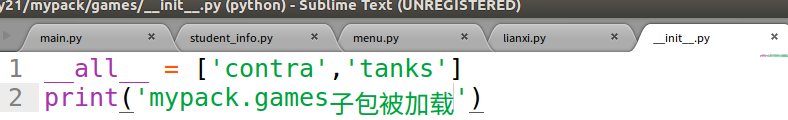
作用：

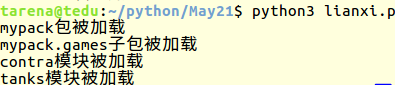
用来记录此包中有哪些子包或模块在用from包import\*语句导入时，是否被导入。

说明：

\_\_all\_\_列表只对from import \*语句起作用

示例：





可以选择性的加载子包内的内容

2 包的相对导入

包的相对导入是指包内模块的相互导入

语法：

from相对路径包或模块import属性或模块名

或

from相对路径包或模块import\*

相对路径：

.代表当前目录

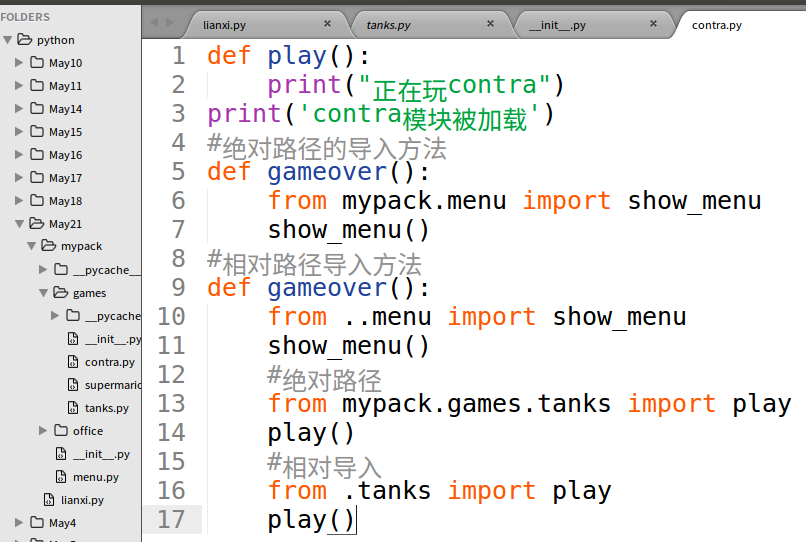
..代表上一级目录

...代表上二级目录

....以此类推

注：相对导入时不能超出包的外部

示例：



包的加载路径：

同模块的加载路径相同

1.当前文件夹

2.sys.path给出的路径

3 异常

异常exception

什么是错误？

错误就是指由于逻辑或语句等导致一个程序无法正常执行的问题

特点：

有些错误是无法预知的

什么是异常？

异常是程序出错时标识的一种状态

当异常发生时，程序不会再向下执行，而转去调用此函数的地方待处理此错误并恢复为正常状态

异常的作用：

1.通知上层调用者有错误产生需要处理

2.用做信号

4 try

try语句的两种用法

try - except

try - finally

try - except语句

语法：

try:

可能触发异常的语句

except 错误类型1 [as 变量1]：

异常处理语句1

except 错误类型2 [as 变量2]：

异常处理语句2

except （错误类型3，错误类型4……） [as 变量3]：

异常处理语句3

……

expect： #可捕获任何类型

异常处理语句other

else：

未发生异常时执行的语句

finally：

最终执行语句

做用：

尝试补获异常，将程序转为正常状态并继续执行

语法说明：

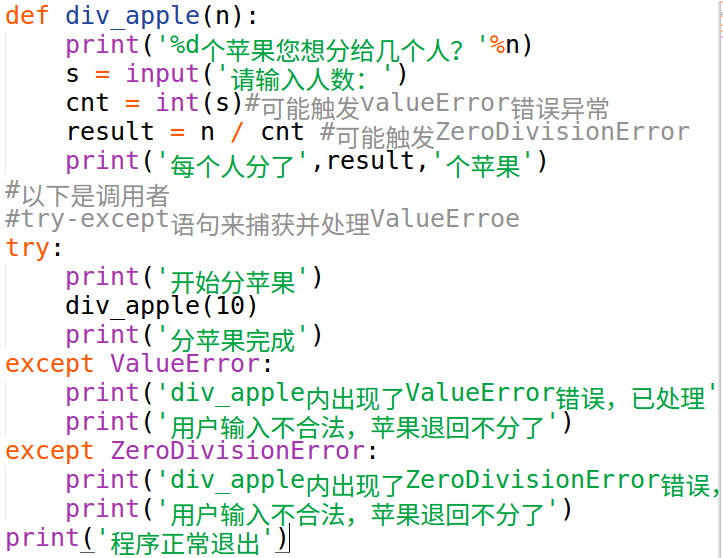
as 子句是用于绑定错误对象的变量，但可以省略不写

except子句可以有一个或多个，但至少要有一个

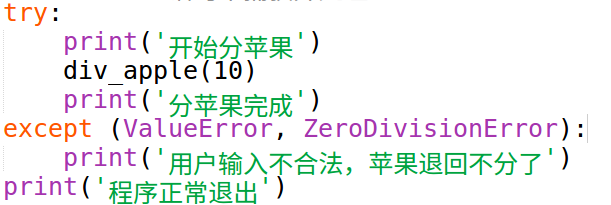
else 子句最多只能有一个，也可以省略不写

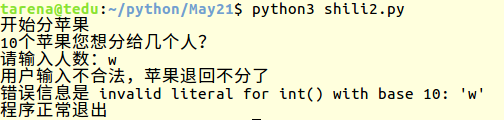
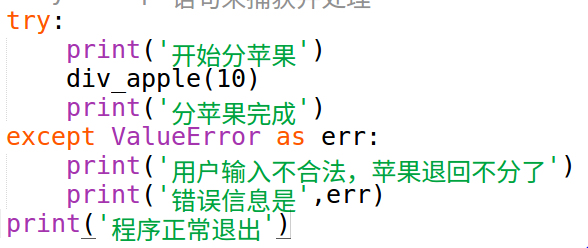
finally子句最多只能有一个，但可以省略不写

示例：



写法二：





| **错误类型** | **说明** |
| --- | --- |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| ValueError | 传入无效的参数 |
| AssertionError | 断言语句失败 |
| StopIteration | 迭代器没有更多的值 |
| IndexError | 序列中没有此索引(index) |
| IndentationError | 缩进错误 |
| OSError | 输入/输出操作失败 |
| ImportError | 导入模块/对象失败 |
| NameError | 未声明/初始化对象 (没有属性) |
| AttributeError | 对象没有这个属性 |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| LookupError | 无效数据查询的基类 |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾  回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError Python | 语法错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| 以下为警告类型 |  |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

练习:

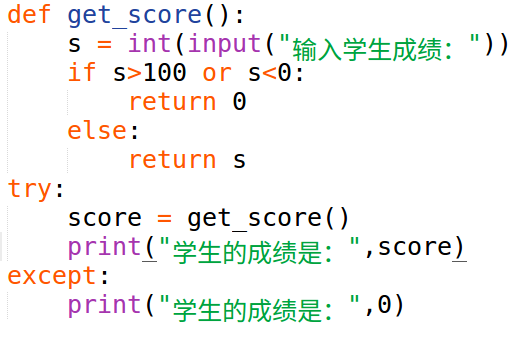
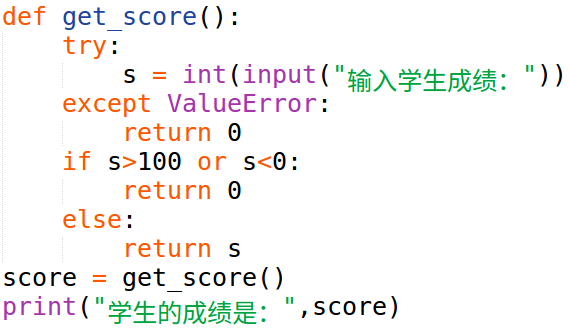
写一个函数get\_score()来获取学生的成绩0~100，如果输入出现异常，则此函数返回0，否则返回用户输入的成绩

def get\_score():

……

score = get\_score()

print(“学生的成绩是：”,score)



**4.1 try-finally语句**

语法：

try:

可能触发异常的语句

finally:

最终语句

说明：

finally子句不可省略

一定不存在except子句

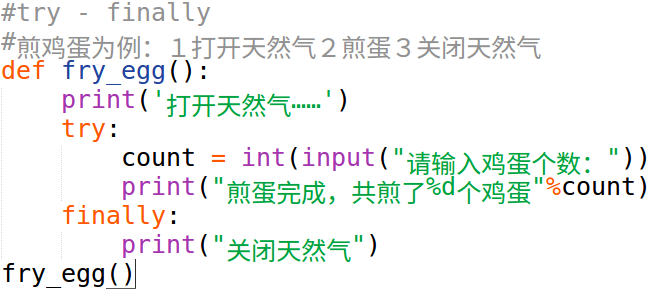
作用：

通常用try-finally语句来做触发异常时必须要处理的事情，无论一场是否发生，finally子句都会被执行

注：

try-finally语句不会改变程序的（正常/异常）状态

示例：



问题：

try-except语句干什么用？

请问错误通知是谁发出的？怎么发出？

**4.2 raise语句**

作用：

触发一个错误，让程序进入异常状态

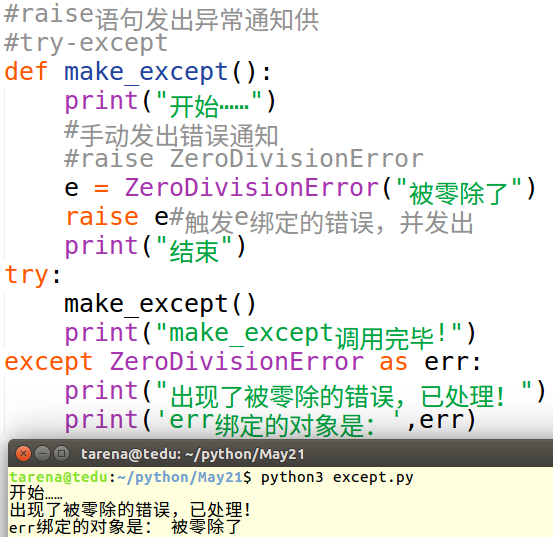
语法：

raise 异常类型

或

raise 异常对象

示例：



示例：



**4.3 assert语句（断言语句）**

语法：

assert 真值表达式，错误数据（通常是字符串）

作用：

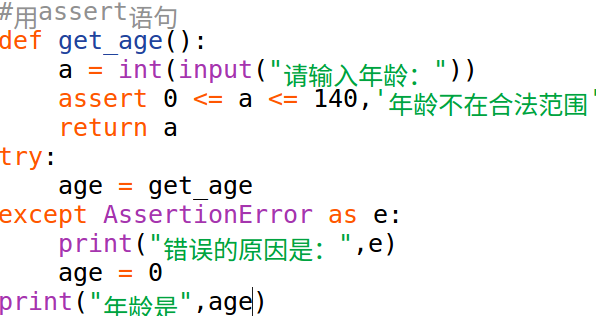
当真值表达试为False时，用错误数据创建一个AssertionError类型的错误，并进入异常状态

类似于：

if 真值表达式 == False

raise AssertionError（错误数据）

示例：



5 小结

接受消息的：

try – except

做必须处理的事情：

try – finally

发错误消息的：

raise语句

assert语句

为什么要用异常处理机制？

原因是：

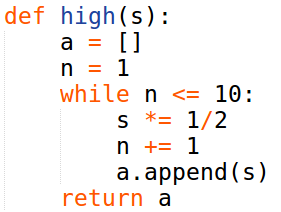
在程序调用层数较深时，向主调用函数传递错误信息需要用return语句层层传递比较麻烦，所以用异常处理机制

练习：

1.一个球100米高空落下，每次落下后反弹高度是原高度的一半，再落下

1）写出程序算出皮球再第10次落地后反弹的高度是？

2）打印出球共经过了多少米的路程



2.分解质因数，输入一个正整数，分解质因数：

如：

输入：90 则打印：90=2\*3\*3\*5

（质因数是指最小能被原数整除的素数（不包括1）