python高级

1、嵌套循环

使用的场景：

- 面试中的笔试题（例如：写一个冒泡循环，遍历某个数据）
- 工作中，需要遍历数据的

学习的内容：

- 九九乘法口诀表
- 排列三角形图案
- 冒泡排序

遍历数据

```
lis=[['a','b','c'],[1,2,3,4,5]]

for i in lis: #2次
    for x in i:
        print(x)
    print('哈哈')

'''
# 第一层for循环：
    第一次运行，i = ['a','b','c']
    第二次运行，i = [1,2,3,4,5]

第二层for循环
    x 第一次：a
    x 第二次：b
    x 第三次：c
    x 第四次：1
    x 第五次：2
    x 第六次：3
    x 第七次：4
    x 第八次：5
'''
```

遍历字典：

```
dic={'one':{1:'壹',2:'贰'},'two':{3:'叁',4:'肆'}}

for k,v in dic.items():
    for k1,v1 in v.items() :
        print(k1)
        print(v1)
```

三角性图案

```
#正序的三角型：
*
**
***
****

for i in range(5):
    for x in range(i+1):
        print('*',end='')        #end=''表示不换行打印
    print()                 #表示会进行换行

#倒序三角
****
***
**
*

for i in range(5):
    for x in range(5-i):
        print('*',end='')        #end=''表示不换行打印
    print()                 #表示会进行换行


#反向正序三角

   *
  **
 ***
****

for i in range(5):
    for _ in range(5-i):
        print(" ",end='')

    for _ in range(i+1):
        print("*", end='')
```

冒泡排序

```
lis=[2,34,5,7,5,7,8,422,437,87]
    # 0 1 2
# 从小到大进行排序
# i=0,1,2,4,5,6,7,8,9
for i in range(len(lis)): #0
    for j in range(len(lis)-i-1):  #range()= 9,8,7,6,5,4,3,2,1
        #j 0,1,2,3,4,5,6,7,8,9
        if lis[j] > lis[j+1]:
            lis[j] , lis[j+1] = lis[j+1] , lis[j]
print(lis)
```

乘法表

```
for i  in  range(1,10):
    for j in range(1,i+1):
        print( f'{j} * {i} = {i*j}',end='\t')
    print()
```

2、函数

- 函数：
  - 已经封装好的功能，在调用的时候使用。print() ,range()
- 创建函数：
  - 取名规则：
    - 不能使用中文和除了下划线意外的符号
    - 函数名不可以使用已有的函数名
    - 不可以使用空格
    - 尽量使用英文，拼音

```
def 函数名():
    函数体


举例：

def car():
    print('我有一辆超级跑车！')

car()   #调用函数
```

没有参数的函数

```
def car():
    print('我有一辆超级跑车！')
    print('我有一辆黄色的超级跑车！')
    print('我有一辆蓝色的超级跑车！')

car()   #调用函数
```

有一个参数的函数

函数可以定义一个参数，在调用函数时必须传入参数来执行

```
def car(col):   #定义一个参数
    print('我有一辆超级跑车！')
    print(f'我有一辆{col}的超级跑车！')   #使用传入的参数
    print(f'我有一辆{col}的超级跑车！')

car('蓝色色')   #调用函数时必须传入参数



def add(num):
    num=num+1
    print(num)

add(6)

-------------------------------------
举例：
    def order(lis):
    for i in range(len(lis)):
        for j in range(len(lis)-i-1):
            if lis[j] > lis[j+1]:
                lis[j] , lis[j+1] = lis[j+1] , lis[j]
    print(lis)


lis=[6,5,4,3,2,1]
order(lis)
```

传入多个参数

```python
def sum_1(num1,num2):
    sum=num1+num2
    print(sum)

sum_1(num2=8,num1=9)     #不需要考虑参数的位置
#传入的参数和位置要和函数的先后位置以及数量要一致
#如果指定参数，则不用考虑参数的位置（num2=8,num1=9）
```

默认或者空参数

```python
def hours(floor1,col='红色',window=None):  #col参数的默认为'红色'
    print(f'我有一栋{col}的房子，它有{floor1}层！,它{window}')

hours('五','黑色') #传入了col的参数 ，结果为：我有一栋黑色的房子，它有五层！,它None
hours('五')  #没有传入col的参数，结果为：我有一栋红色的房子，它有五层！,它None
hours('五','黑色','有窗')     #传入了window参数，结果为：我有一栋黑色的房子，它有五层！,它有窗

空参数None 如果有传入，则使用传入的参数，如果没有传入，则使用None
注意：
    默认参数要在非默认参数的后面
```

不定长参数1

不确定传入的参数的数量，使用一中省略的写法

带调用时实现直接传入变量参数

‘*’代表不定长

args 传入的参数

传入的参数返回的结果是元组数据

```python
def post1(*args):
    print(args)
    print(type(args))

post1('张三','李四','王五')

#('张三', '李四', '王五')
#<class 'tuple'>
```

不定长参数2

不定长参数：**kwargs

第一个‘ * ’代表的是key

第二个 * 代表的是value

args 代表的是传入的参数

得到的是一个字典类型的数据

```
def mingxing(**kwargs):
    print(kwargs)
    print(type(kwargs))

mingxing(name='刘德华',age='50',hight='180')    #调用函数并且传入参数

# {'name': '刘德华', 'age': '50', 'hight': '180'}
# <class 'dict'>
```

return 和 print的区别

```
def A(num):
    return num+1
# print()将结果打印到控制台中，在调试的时候可以通过print()来进行调试
#return是内部运算，返回结果，返回的结果可以给其他程序调用，但是不会打印到控制台中
A(5)
print(A(5)+1)  #函数返回结果后在进行计算，如果函数中没有 return，则无法进行结果计算
```

3、深浅拷贝

浅拷贝：拷贝的是这个变量或值的id,如果这个值被改变了，拷贝的后的数据也会同步发生改变

深拷贝：拷贝之后的数据不会因为原来的数据发生改变而改变

```
import copy     #导入copy库

li=[1,2,3,[4,5,6],7,8]
cp=li.copy()        #浅拷贝
dcp=copy.deepcopy(li)   #深拷贝

li[3][0]=88     #拷贝之后对这个数据进行了修改

print(cp)       # 浅拷贝的值发生了改变，[1, 2, 3, [88, 5, 6], 7, 8]
print(li)       #
print(dcp)      #深拷贝的值没有发生改变，[1, 2, 3, [4, 5, 6], 7, 8]
```

4、类

对象

对象：根据对实物的理解，对象在代码中是虚拟出来的概念。在编程对象中，对象的属性是描述对象的特点，对象的行为可以是方法，方法就是对应函数

类

面向对象需要的基础，类是属性和方法的集合。类是对象的模板，对象是类的实例

```
class dog():
    color=None
    size=None
    weight=None

# 实例化
Dog=dog()
Dog.color='红色'
Dog.size='100cm'
Dog.weight='10KG'
print(f'这是一只颜色为{Dog.color},长度为{Dog.size},体重为{Dog.weight}的狗')

Dog1=dog()
Dog1.color='黑色'
Dog1.size='10cm'
Dog1.weight='1KG'
print(f'这是一只颜色为{Dog1.color},长度为{Dog1.size},体重为{Dog1.weight}的狗')
```

类的方法和属性

```
class 类名():
    pass
```

类和方法举例

```
class dog():    #创建类
    color=None      #类的属性
    size=None       #类的属性
    weight=None       #类的属性

    def run(self):      #类的方法
        print(f'这是一只颜色为{self.color},长度为{self.size},体重为{self.weight}的狗,
它跑起来了')

    def eat(self):      #类的方法
        print(f'这是一只颜色为{self.color},长度为{self.size},体重为{self.weight}的狗,
它吃起来了')

# 实例化
Dog=dog()       #实例化类
Dog.color='红色'
Dog.size='100cm'
Dog.weight='10KG'
```

```
Dog.run()        #调用类的方法
Dog.eat()        #调用类的方法
```

构造函数

构造函数的作用：在运行类的时候，会优先调用这个构造函数

```
class dog():    #创建类
    def __init__(self,color,size,weight):
        self.color=color        #将参数实例化
        self.size=size
        self.weight=weight
        #在运行类的时候，会先调用初始化（构造函数）这个函数有哪些参数需要传入的，在调用的时候就需要传入参数
    def run(self):       #类的方法
        print(f'这是一只颜色为{self.color},长度为{self.size},体重为{self.weight}的狗,它跑起来了')

    def eat(self):       #类的方法
        print(f'这是一只颜色为{self.color},长度为{self.size},体重为{self.weight}的狗,它吃起来了')

Dog=dog('黄色','100cm','100KG')
Dog.run()
```

一起皆是对象

在python里，所有的东西都可以当成一个对象

举例

```
class V :   #定义一个类
    a=100

lis=[1,2,3,4]
lis.append(V)   #将类当成一个对象加入到列表中
print(lis)
```

购物车和商品案例

```
'''
写一个购物车，商品功能
商品：
    商品名称，价格，商品ID
购物车：
    存放/添加商品
    总价
    单价
    商品数量
```

```
    删除或减少商品数量
'''

class Goods:

    goods_id=0

    def __init__(self,goodsname,price):
        self.goodsname=goodsname
        self.price=price
        self.goods_id +=1
        Goods.goods_id = self.goods_id  # 修改类属性


class Cart:
    '''
    购物车：
    存放 / 添加商品
    总价
    单价
    商品数量
    删除或减少商品数量
    '''
    def __init__(self):
        self.goods_list=[]
        self.goods_coount={}

    def add(self,goods,num=1):
        '''
        数据校验：对商品是否存在做校验，对传入的商品的数量（int）做格式校验
        需要对购物车进行判断：
            如果商品已经存在购物车中，则对商品数量进行增加
            如果商品不在购物车中，添加商品和商品数量
            加入购物车时，商品应默认一个
        '''

        if goods in self.goods_list:
            self.goods_coount[goods.goods_id] += num
        else:
            self.goods_list.append(goods)
            self.goods_coount[goods.goods_id] = num      #键为商品ID，数量为num,通过
键值对来保存商品数量


    def remove(self,goods,num):
        '''
        传入的参数格式校验
        根据商品ID来删除或减少商品的数量
        判断：
            商品是否在购物车中
            如果商品的数量为0，要将商品从列表中删除

        '''
        if goods not in self.goods_list:     #判断商品是否在购物车中
            print('商品不在购物车中')
```

```
        else:
            self.goods_coount[goods.goods_id] -= num     #如果存在购物车中，则减少商品的数量

        if self.goods_coount[goods.goods_id] <= 0 :    #如果商品数量小于或等于0时，需要从列表和字典中删除

            del self.goods_coount[goods.goods_id]
            self.goods_list.remove(goods)

    def count_price(self):  #将所有商品的价格求和存放在count_price里面
        count_price=0
        for goods in self.goods_list:        #遍历整个商品列表
            count_price += goods.price * self.goods_coount[goods.goods_id]  #计算商品总价，这里是商品单价乘以数量
        return count_price

    def show(self):
        print('品名','\t数量','\t单价','\t小计')
        for goods in self.goods_list:
            min_count=self.goods_coount[goods.goods_id] * goods.price
            print(goods.goodsname,f'{self.goods_coount[goods.goods_id]}',f'\t{goods.price}' ,f'\t{min_count}')      #商品名称

        print('*'*30)
        print(' '*15,'总价：',self.count_price())


iphon13=Goods('iphon13',12000)
g2=Goods('兰博基尼',40000)
g3=Goods('联想电脑',3000)

cart=Cart()
cart.add(iphon13,10)
cart.add(g2,6)
cart.add(g3,6)


a=cart.goods_coount
print(a)
cart.show()
```

类的继承

概念：继承就是让类和类之间转化为父子关系，子类可以直接访问（调用）父类的静态属性和方法

继承的规则：

- 子类继承父类的成员的变量和成员方法
- 子类能继承父类的构造方法,可以继承析构方法
- 子类不能删除父类方法，但是可以从新定义父类方法
- 子类可以增加自己的方法
- 子类重写了父类的构造函数，在实例化的时候会使用子类的构造函数

举例:

```
class Dog:
    color='黄色斑纹'
    size='小型犬'
    def eat(self):
        print(f'{self.color}的大狗子在吃饭')

class Dog_son(Dog):          #继承是将父类的名字写在括号中
    def new_eat(self):
        print('的小狗子在吃饭')

dog=Dog_son()
print(dog.color)         #子类继承了父类的属性
print(dog.size)
dog.eat()                #子类继承了父类的方法
dog.new_eat()            #子类自己的方法
```

类的重写

子类的方法如果和父类的方法名称产生重复，则会重写父类方法

```
class Dog:
    color='黄色斑纹'
    size='小型犬'
    def eat(self):
        print(f'{self.color}的大狗子在吃饭')

class Dog_son(Dog):          #继承是将父类的名字写在括号中
    def new_eat(self):
        print('的小狗子在吃饭')

    def eat(self):           #与父类的方法重名
        print('小狗子吃饭')

    def eat_father(self):    #子类想保留父类的方法，通过定义一个新的函数，来调用父类的方法
        Dog.eat(self)

dog=Dog_son()
print(dog.color)         #子类继承了父类的属性
print(dog.size)
dog.eat()                #子类继承了父类的方法
dog.eat_father()
# dog.new_eat()            #子类自己的方法
```

多态：

5、pymysql

python可以使用pymysql库去连接数据库进行操作

使用场景：

实现数据闭环（测试产生的数据，在结束测试的时候对产生的数据执行删除）

自动化测试的时候需要去数据库里面查找数据

安装pymysql库：

- 点击左上角 'File' --> 选择 'seting' -->再选择Project: xxx --> 再选择右上角的 ' + '号
- 在上一步弹出框中搜索： pymysql --> 选中 pymysql 2 -->再点击左下角的 install Package 下载完成即可

第一步：建立数据库连接，

第二步：再通过连接创建一个游标，

第三步：通过游标来执行sql语句

拿到数据库服务器的地址，账号，密码

```
import  pymysql

db=pymysql.connect(host='106.52.182.140' ,
                   user='root' ,
                   passwd='Cxy199807.' ,
                   database='gz2242' ,
                   charset='utf8')

cursor=db.cursor()      #创建了一个游标
```

使用游标执行语句

```
cursor.execute('select * from dep')     #传入sql语句
```

查询：

```
data=cursor.fetchone()      #返回第一条
print(data)

cursor.fetchmany(num)   #返回num条数据
举例：
    data1=cursor.fetchmany(5)   #返回num条数据
    print(data1)

cursor.fetchall()       #返回所有的查询结果
```

创建表

```
cursor.execute(建表语句)
举例：
    cursor.execute('create table g2242(id int primary key , name varchar(5) ,
class int)')
```

插入数据

```
#插入数据
cursor.executemany(sql语句,插入的数据)


举例：


connect = pymysql.connect(host='106.52.182.140',
                      user='root',
                      passwd='Cxy199807.',
                      database='gz2242',
                      charset='utf8')        #建立数据库的连接对象，然后赋值给db变量
cu=db.cursor()      #新建游标，因为在pymysql里面是需要通过游标的方式去执行sql语句

#插入数据
sql ='insert into dep(id ,name) values (%s,%s)'
data=[
    (10001,'张三'),
    (10002,'李四'),
    (10003,'王五')
      ]
cu.executemany(sql,data)    #插入数据使用的是这个方法，会自动将sql和data进行拼接执行
connect.commit()     #通过连接去提交的
```

删除数据：

```
import  pymysql  #需要导入pymysql这个库才可以使用连接数据库的方法和属性

connect = pymysql.connect(host='106.52.182.140',
                      user='root',
                      passwd='Cxy199807.',
                      database='gz2242',
                      charset='utf8')        #建立数据库的连接对象，然后赋值给db变量
cu=connect.cursor()      #新建游标，因为在pymysql里面是需要通过游标的方式去执行sql语句

#删除数据
sql='delete from dep where id=%s'       #删除语句
id=1        #删除的id
cu.execute(sql,id)  #会拼接并执行
connect.commit()    #提交改动
```

关闭数据库连接和关闭游标

```
import  pymysql  #需要导入pymysql这个库才可以使用连接数据库的方法和属性

connect = pymysql.connect(host='106.52.182.140',
                      user='root',
                      passwd='Cxy199807.',
                      database='gz2242',
                      charset='utf8')        #建立数据库的连接对象，然后赋值给db变量
cu=connect.cursor()      #新建游标，因为在pymysql里面是需要通过游标的方式去执行sql语句



cu.close()      #关闭游标
connect.close() #关闭连接
```

6、xlrd

通过xlrd这个库来读取excel表格的内容

读取步骤：

- 打开表格
- 选择读取的工作页
- 根据需要返回工作页中的内容，可以一行读取，或者一列，在或者单元格

打开表格

可以在路径的最前面加上r，r表示格式化后面的字符为字符串

```
xl = xlrd.open_workbook(r'C:\Users\36545\Desktop\2242课程表.xlsx')
    #打开表格，括号年内填入excel表格的路径
```

选择工作页

```
#选择工作页
# 通过赋值的变量去打开工作页
# 获取整个表格的所有工作页

xl.sheet_names()          #获取到整个表格的所有工作页的名称
# table=xl.sheet_by_name('num3')    #按照工作页的名称来选择
table=xl.sheet_by_index(1)      #通过工作页的索引来选取工作页
# xl.sheets()[]         #通过下标来选择工作页
```

读取行

```
#读取行
num=table.nrows    #获取该表的所以有的有效行的数量
print(num)       #打印返回的行数

data=table.row_values(0,start_colx=0,end_colx=3)      #返回该行所选中的数据，
    #   rowx   ：对应的行
    # start_colx=0    :开始的列
    # end_colx=None   ：结束的列

print(data) #打印

data1=table.row_len(2)  #返回该行中有效的数据长度，这里计算的长度是整个表中行数最长的数量
print(data1)
```

读取列

```
nco=table.ncols  #获取所有的有效列数
print(nco)

col=table.col_values(1,1)
print(col)
# colx ： 读取的列的索引，第一列的索引是0
# start_rowx=0 ：读取的开始行的索引，第一行是0
# end_rowx=None ：  读取结束行的索引，默认是全部
```

读取单元格

```
# table.cell(rowx=行,colx=列)      #返回的是对应一个单元个的对象

data3=table.cell(2,1)
print(data3)
print(type(data3))

# table.cell_value(rowx=行,colx=列)        #返回的是单元格里面的值
data4=table.cell_value(2,1)
print(data4)
print(type(data4))
```

7、random

准确来说生成的是伪随机数。random会根据当前的操作系统的时间作为随机数的种子，所以可以保证生成的随机数一般不会重复。

```
import random

#产生随机的浮点数，产生随机数的范围：0.0 - 1.0
flo = random.random()
print(flo)

#产生1到100之间的随机整数，包括1和100
ran1 = random.randint(1, 100)
print(ran1)

#从序列中随机抽取一个元素
li=[1,2,3,4,5,6,'a','b','c','今','天']
ran2 = random.choice(li)
print(ran2)

#从序列中抽取出K个元素，抽取出来的元素可能会重复
li_1 = (1, 2, 3, 4, 5, 6, 'a', 'b', 'c', '今', '天')
ran3 = random.choices(li_1, k=2)  #从序列中随机抽取2个元素
print(ran3)

#从序列中抽取元素，但是每次抽取的元素不会重复，例如：第一次运行抽取两个元素，这两个元素不会是同一个
li_2 = ( 'a', '今', '天')
ran4 = random.sample(li_2, k=2)      #抽取两个随机元素
print(ran4)

#将一个序列随机打乱，注意：这个序列不能是只读
lis_3=[1,2,3,4,5,6,'a','b','c','今','天']
ran5 = random.shuffle(lis_3)
print(lis_3)        #注意这里打印的是lis_3
```

8、os

os是操作系统文件的接口

```
import os

#获取当前工作文件的路径,返回的是字符串类型
getwrok=os.getcwd()
print(getwrok)

#返回对应文件下的所有子文件和目录
#返回的是列表，包括文件类型
# os.listdir(填入需要查找的目录的位置)
dir1=os.listdir(r'C:\Users\36545\Desktop\简历')
print(dir1)

#创建指定目录
os.mkdir(路径)
os.mkdir(r'D:\pycode\2242\aa')
os.mkdir(os.getcwd()+'aa') # 在当前工作文件的目录下创建一个aa目录


#判断是否为目录或文件
os.path.isdir()      #判断是否为目录
os.path.isfile(路径)    #判断是否为文件

f=os.path.isfile(os.getcwd())
print(f)

#返回分隔符
os.sep  #返回当前操作系统的分隔符


#路径拼接
os.path.join(路径1,路径2)
# 举例
j=os.path.join(os.getcwd(),'aa','bb','cc')
print(j)  #D:\pycode\2242\aa\bb\cc

#返回路径，但是不包括最后一个目录
x=os.path.dirname(os.getcwd())
print(x)


#返回最后一级的名称,如果是文件路径，会返回文件名称
b=os.path.basename(os.getcwd())
print(b)
```

9、json

json的全称为，是一种轻量级的数据交互格式

基本所有的语言都支持json数据格式

json数据和python的dict比较相似，但是json并不是字典

学习的内容：

学习数据之间的转换

json-->python

- json.loads() # 将json的数据转换成python的数据结构

  json模块在转换的时候，会根据你的数据则字符串自动转换为最合适的python数据类型

```
举例：
import  json  #导入json库
import requests
get_json=requests.get(url='http://106.52.182.140/upload/goods.php?
act=price&id=140&attr=&number=1&1652840451383383') #使用requests库从网站中获取到
json数据

js_data=get_json.text   #返回接内容中有json数据

print(js_data)      #打印json数据
print(type(js_data))    #判断格式，json格式在python中的数据格式是str
print(js_data[3:15])    #字符串只能使用切片去获取数据


py_data=json.loads(js_data,)    #将json格式转换为python格式
print(py_data)          #打印转换后的python数据
print(type(py_data))    #打印转换后的数据类型



参数：
    parse_int = float   #把json数据中int类型的数据转换成float格式的python数据类型
    parse_float  = int  #把json数据中int类型的数据转换成float格式的python数据类型
    encoding=   #编码
```

- json.load() # 将json数据的文件转换成python的数据结构，使用方法和json.loads()基本一样

python-->json

- json.dumps() #将python的数据结构转换成json数据

```
dic={1:'A',2:'b','一':'壹','er':'二'}
# dis1={"err_msg":"","result":"\uffe5250\u5143","qty":1}

js1=json.dumps(dic,indent=0)
```

```
print(js1)
print(type(js1))

参数：
ensure_ascii=True \ False
        #编码开关,出现乱码的时候可以解决

skipkeys=True \ False,
        #默认是False，如果dict的键不是python的基本数据类型
（str,int,float,bool,None,unicode,long），设置是False时，会有报错

indent=None
        #值需要填入int类型的数据，会根据数据格式进行缩进
```

- json.dump() #将python数据结构的文件转换成json数据文件，使用方法和json.dumps() 基本一样

10、捕获异常

程序在运行的时候，如果python的解释器遇到的一个错误，会停止程序的执行，并且提示一些错误信息，这就是异常。我们在程序开发的时候，很难将所有的特殊情况都进行处理，可以通过捕获异常，对突发事情做集中出具，从而保证的程序的稳定性

第一种

```
写法1：

try:
    可能出现异常的代码
except 异常类型 :              #捕获异常类型为Exception的错误
     针对这个异常需要执行的操作

写法2：

try:
    可能出现异常的代码
except   异常类型   as   e :         #捕获到错误后，将异常信息赋值给e,这个e是自定义的变量
     针对这个异常需要执行的操作

异常类型：
     代码出现的异常的类型如果和捕获的类型一致，则会被捕获，否则不会被捕获
     Exception    --> 异常类型的一个基类
     NameError    -->  名称异常
     TypeError    -->  类型异常


举例：

try:
```

```
    num=int(input('请输入数字：'))
except Exception as e :
    print(e)
    print('请输入数字！')

print(123)
```

第二种：

不管是否异常，都会执行finally里面的代码。

```
try:
   可能出现异常的代码
except 异常类型 :            #捕获异常类型为Exception的错误
    针对这个异常需要执行的操作
finally:
    最后都需要执行的代码


举例：
try:
    num=int(input('请输入数字：'))
except Exception as e :
    print(e)
    print('请输入数字！')
finally:
    print('这个语句必定会被执行！')
```

第三种：

如果try的代码里面出现异常，后面的代码不会被继续执行，如果没有异常，在会执行else里面的二代码

```
try:
   可能出现异常的代码
except 异常类型 :            #捕获异常类型为Exception的错误
    针对这个异常需要执行的操作
else:
    没有异常执行的代码


举例：
try:
    num=int(input('请输入数字：'))
except Exception as e :
    print(e)
    print('请输入数字！')
else:
    print('这个语句是否会被执行！')
```

11、抛出异常

当代码没有语法错误是，python是不会处理异常的。但是有些业务场景下，是需要抛出异常的，可以使用raise方法来主动抛出异常。

```
格式：
raise Exception('需要抛出的异常提示内容')

举例：

phon_num = input('手机号:')
if len( phon_num) != 11:     #如果phon_num的长度不等于11位
    raise Exception('手机号的长度要等于11位')      #抛出一个Exception异常

print('123')
```

12、断言

assert ，断言在软件开发中是一种常用的调试方式。assert就是在程序中的一条语句，它对程序进行检查，一个正确的程序的boolean表达式的值必须为True，如果该值为False ,说明程序已经处于不正确的状态下，系统将给出警告并且退出。

assert 一般是用来判断程序中关键条件的正确性，比如：一辆汽车是否能够正常行驶的重要条件是轮胎的状态，如果轮胎爆胎了，不能在继续行驶。就像是预防错误一样，为了防止继续执行代码可能造成损失等等所做的一种防范措施。

```
# 格式
assert 条件 ，引发断言后需要抛出的内容

# 举例
phon=input('手机号：')
assert (len(phon) == 11) ,'输入的手机号的长度要等于11位'

print(123) #assert 语句如果出现异常，则后面的代码都不会再继续执行
```