# Python简介

Python是一种面向对象的编程语言，跨平台，可以在不同操作系统上运行。Python提供了非常全面的代码基础库，覆盖了网络、文件、GUI、数据库、文本等大量内容，被形象地称作“内置电池”。用Python开发，许多功能不必从零编写，直接使用现成的即可。除了内置的库外，Python还有大量的第三方库可供使用。所以Python程序看上去总是简单易懂，可读性较好，上手比较容易。Python的哲学就是简单优雅，尽量写容易看明白的代码，尽量写少的代码。

当然优点颇多的Python，也有其缺点。第一个缺点是运行速度，与C语言等编译型语言相比，Python是解释型语言，代码在执行时是一边编译一边执行所以运行时比较慢。第二个缺点是保密性，发布Python程序，实际上就等于发布Python的源代码，不像C语言、Java语言发布的时候是发布编译之后的机器码。

# Python安装

首先根据Windows版本（64位还是32位）从Python的官方网站下载Python对应的64位安装程序或32位安装程序。



然后，运行下载的exe安装包，记得需要勾选将Python添加到环境变量变量中，如果不勾选则需要自己手动在环境变量中添加Python。



安装成功后，打开命令提示符窗口，输入Python，如出现以下信息则表示Python安装成功且环境变量配置正确。



# Python解释器

在编写Python代码时，会得到一个以.py扩展名的文本文件，要运行代码需要Python解释器。Python解释器将源代码转化为字节码，如果Python进程在运行的环境中具有写入的权限，那么将会生成一个.pyc的文件，字节码保存在.pyc文件中（类似java代码文件编译成的class文件）。每次执行Python代码时，如果Python文件改动过那么会重新编译生成（通过检查Python源文件与字节码文件的时间戳）。如果不具备写入的权限，那么字节码将在内存中生成，执行完毕后会被丢弃。Python代码转化成字节码后会被发送到PVM（Python虚拟机）执行，PVM是Python能够跨平台的核心。

1. CPython

CPython是标准Python，也是其他Python编译器的参考实现。CPython由C编写，将Python源码编译成CPython字节码，由虚拟机解释执行。

（2）JPython

Jython在JVM上实现的Python，由Java编写。Jython将Python源码编译成JVM字节码，由JVM执行对应的字节码，因此能很好的与JVM集成。

# Python输出与输入

Python中用print()来打印输出，用print()可以打印字符串，字符串需要用引号括起来，引号可以是单引号或者双引号，例如print(‘hello world!’)，结果输出hello world!。print()也可以打印数值，或者计算结果，例如print(33.33)：结果输出33.33，print(3\*5)：结果输出15。print()函数可以打印多个输出，每个输出用逗号隔开，逗号输出为空格。例如print(‘3+5’,’=’,3+5)，结果输出3+5 = 8。

Python中用print()输出结果，同时提供了一个input()处理用户输入。例如：

length=float(input(‘请输入正方形的边长：’))

print(‘正方形的面积为：’,length\*length)

input()函数返回值是str型，如果输入的值作为数值型的参数，那么需要进行强制转换。执行这段代码首先会提示让输入一个正方形的边长，然后在输入正方形的边长之后，会输出正方形的面积。

# Python基础

## Python编写规范

Python的语法比较简单，写出来的代码就像下面的样子：

# Python编写规范  
a=100  
if a>=0:  
 print(a)  
else:  
 print(-a)  
# Python类（class）  
class TEST():  
 def test1(self):  
 print('方法1')  
 def test2(self):  
 print('方法2')

以#开头的语句是注释，可以是任意内容，解释器会忽略掉注释。Python中，对于类定义、函数定义、控制语句、异常处理语句等，行尾的冒号和下一行的缩进，表示下一个代码块的开始，而缩进的结束则表示此代码块的结束。Python中实现对代码的缩进，可以使用空格或者Tab键实现。但无论是手动敲空格，还是使用Tab键，通常情况下都是采用4个空格长度作为一个缩进量（默认情况下，一个Tab键就表示4个空格）。缩进让代码格式化，增加代码的可读性。Python对代码的缩进要求非常严格，同一个级别代码块的缩进量必须一样，否则解释器会报 SyntaxError 异常错误。此外Python程序是大小写敏感的，如果写错了大小写，程序也会报错。

## 数据类型和变量

Python能够直接处理的数据类型有以下几种：

1. 整数

Python可以处理任意大小的整数，当然包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：1，100，-8080，0，等等。

1. 浮点数

浮点数也就是小数，浮点数可以用数学写法，如1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把10用e替代，1.23x109就是1.23e9，或者12.3e8，0.000012可以写成1.2e-5，等等。

（3）字符串

字符串是以单引号或双引号括起来的任意文本，比如'abc'，"中国"等等。引号本身只是一种表示方式，不是字符串的一部分，因此字符串'abc'只有a，b，c这3个字符。如果引号本身也是一个字符，那必须用引号括起来，比如"I'm OK"包含的字符是I，'，m，空格，O，K这6个字符。

如果字符串内部既包含单引号又包含双引号，就需要用转义字符\来标识，比如'I\'m \"OK\"!' 表示的字符串内容是I'm "OK"!。转义字符\可以转义很多字符，\t表示制表符，字符\本身也要转义，\\表示的字符就是\。\n表示换行，为了可读性比较好，Python一般用’’’...’’’表示多行内容。但如果本身字符串就带有\，例如文件的路径信息，这时候就不需要进行转义，Python中在字符串前加r表示字符串不转义，例如r”E:\n\t\test”表示的字符串内容为E:\n\t\test。

（4）布尔值

布尔值和布尔代数的表示完全一致，一个布尔值只有True、False两种值，要么是True，要么是False。在Python中，可以直接用True、False表示布尔值（请注意大小写），也可以通过布尔运算计算出来：print(True)：True，print(False)：False，print(5>3)：True，print(5>10)：False，print(5+10==15)：True。

布尔值可以用and、or和not运算。and运算是与运算，只有所有都为True，and运算结果才是True，例如print(3>5 and 10>8)：False。or运算是或运算，只要其中有一个为True，or运算结果就是True，例如print(3>5 or 10>8)：True。not运算是非运算，它是一个单目运算符，把True变成False，False变成True，例如print(not 3>5)：True。

1. 空值

空值是Python里一个特殊的值，用None表示。None不能理解为0，因为0是有意义的。

（6）变量

变量的概念基本上和初中代数的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意数据类型。变量在程序中就是用一个变量名表示了，变量名必须是大小写英文、数字和\_的组合，且不能用数字开头，比如：a = 1，变量a是一个整数；t\_007 = 'T007'，变量t\_007是一个字符串；Answer = True，变量Answer是一个布尔值True。在Python中，等号是赋值语句，可以把任意数据类型赋值给变量。

（7）格式化

%运算符就是用来格式化字符串的，在字符串内部，有几个%?占位符，后面就跟几个变量或者值，顺序要对应好。如果只有一个%?，括号可以省略。常见的占位符有：%d——整数、%s——字符串，%f——浮点数(四舍五入保留6位小数)。例如’%d11’%100结果为10011，’%scde’%’ab’结果为abcde，’%f456’%1.2345678结果为1.234568456，’%d456%sdef%f’%(123,’abc’,1.234567)结果为123456abcdef1.234567。如果字符串里面的%是一个普通字符怎么办，就需要进行转义，用%%来表示一个%。另一种格式化字符串的方法是使用字符串的format()方法，它会用传入的参数依次替换字符串内的占位符{0}、{1}......，例如’{0}456{1}def{2}’.format(123,’abc’,1.2)结果为123456abcdef1.2。

## 列表与元组

（1）list列表

list是Python的一种有序列表，列出班里所有同学的名字，就可以用一个list表示：classmates=[‘小明’,’小红’,’小刚’]，变量classmates就是一个list，使用len函数可以获取列表的元素个数：len(classmates)=3。可以使用索引来访问列表的元素，注意索引从0开始classmates[0]表示第一个元素：小明，最后一个元素的索引为len(classmates)-1，如果索引超出范围Python会报一个IndexError错误。此外，可以利用-1，-2，-3来获取列表倒数几个的元素。

list是一个可变的有序表，可以往list中追加元素到末尾：classmates.append(‘小黄’)。也可以把元素插入到指定的位置，比如索引号为1的位置：classmates.insert(1,’小蓝’)。要删除list末尾的元素，用pop()方法：classmates.pop()，如果想要删除指定位置的元素，则使用pop(i)，i为索引。要把某个元素替换成别的元素，可以直接赋值给对应的索引位置：classmates[1]=’小白’。列表中的元素可以也是列表，例如classmates=['小白', ['小红', '小蓝'], '小明', '小刚']，classmates[1][1]结果等于小蓝。此外，可以定义一个空的列表，classmates=[]

（2）tuple元组

Python的另一种有序列表叫元组：tuple。tuple和list非常类似，但是tuple一旦初始化就不能修改，比如同样是列出同学的名字：classmates=(‘小明’,’小红’,’小刚’)。classmates这个tuple不能变，它没有append()，insert()这样的方法。其他获取元素的方法和list是一样的，可以正常地使用classmates[0]，classmates[-1]，但不能赋值成另外的元素。在定义tuple时需要注意的是当tuple只有一个元素时，需要在元素后面加上逗号，以此来区分小括号，例如classmates=(‘测试’,)。但有一些特殊的情况tuple是可以变的，比如tuple中的元素包含列表时，列表中的元素时可以改变的，例如classmates=(‘小明’,[’小红’,’小蓝’],’小刚’)，classmates[1]是一个列表，可以进行append、pop、insert、替换元素邓操作。所谓的“不变”是说，tuple的每个元素，指向永远不变。即指向'a'，就不能改成指向'b'，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的。

## 字典与集合

1. dict字典

Python内置了字典：dict的支持，dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度，例如{'小明':99,'小红':92,'小刚':95}就是一个典型的字典。dict内部存放的顺序和key放入的顺序是没有关系的，所以不能通过索引的方式对字典进行操作。

创建字典有以下常用方法：

1.用等号创建

dict = {'name': 'Tom', 'age': 18}

2.使用dict()和zip()利用已有数据创建

keys = ['a', 'b', 'c']

values = [1, 2, 3]

dict = dict(zip(keys, values))

3.创建空字典

dict = dict()

4.以给定内容为键，创建值为空的字典

dict = dict.fromkeys(['name', 'age'])

可以通过字典的key快速找到其value值，例如chengji[‘小明’]的结果为99。将数据存入字典也可以通过key放入，例如chengji[‘小蓝’]=90。由于一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值冲掉。也可以通过字典的setdefault(key,value)方法往字典里面添加条目，如果键已经存在则添加失败，例如chengji.setdefault(‘小蓝’,95)。当查询的key值在字典中不存在时，dict就会报错。可以通过in来判断key值是否存在在字典中，例如’小蓝’ in chengji，返回的值是一个布尔型。也可以通过dict的get()方法，如果key不存在，返回的结果为None或者自己定义的值，如果存在则返回对应的value值，例如chengji.get(‘小明’,-1)。

要删除一个key，可以使用del、pop(key)方法，对应的value也会从dict中删除，区别在于pop(key)方法会返回对应key的value值，而del方法没有返回值，用法为del(chengji[‘小明’])，chengji.pop(‘小明’)。通过字典的items()方法可以返回键值对组成元组作为元素的列表，通过字典的keys()方法可以返回键的列表，通过字典的values()方法可以返回值的列表，通过字典的clear()方法删除字典的所有键值。update()方法将另一个字典中的键值对添加到当前字典中，若键已存在，则把其对应的值修改为新的字典中相应的值。例如：dict1={‘a’:1,’b’:2,’c’:3}，dict2={‘d’:4,’e’:5}，dict1.update(dict2)结果为{‘a’:1,’b’:2,’c’:3,‘d’:4,’e’:5}。

（2）set集合

set和dict类似，也是一组key的集合，但不存储value。key不能重复，所以在set中，没有重复的key。set可以使用{}或者set()函数进行创建，创建空集合必须set()方法。

使用set(str)方法的方法创建集合会把每字符串每一个字符拆分成元素，例如set(‘abcd’)创建的集合为{‘a’,’b’,’c’,’d’}。也可以通过set(list\tuple)来创建集合，例如set([‘a’,’b’,’c’,’d’])创建的集合为{‘a’,’b’,’c’,’d’}，而使用{}是直接定义一个集合。

通过集合的add(key)方法可以往集合中添加元素，例如set.add(‘e’)，如果集合已经存在元素e，那么添加将无效。使用update方法也可以往集合中添加元素，语法为set.update(x)，x可以为多个，用逗号隔开，参数可以是列表、元组等。通过集合的remove(key)方法可以将集合中的元素删除，例如set.remove(‘a’)，如果元素不存在则会报错。另外一种删除元素方法discard(key)，元素不存在的话也不会报错。此外，还有清空集合的函数clear()。

集合与集合之间也可以进行操作，例如set1 - set2(集合1中含有的元素且集合2中不含有)，set1|set2(集合1与集合2所有元素)，set1&set2(集合1与集合2共同包含的元素)，set1^set2(集合1与集合2不同时包含的元素)。

## 条件判断

计算机之所以能做很多自动化的任务，因为它可以自己做条件判断。比如，输入用户年龄，根据年龄打印不同的内容，在Python程序中，用if语句实现：

age = 20  
 if age >= 18:  
 print('your age is', age)  
 print('adult')

根据Python的缩进规则，如果if语句判断是True，就把缩进的两行print语句执行了，否则，什么也不做。也可以给if添加一个else语句，意思是，如果if判断是False，不要执行if的内容，去把else执行了：

age = 3  
 if age >= 18:  
 print('your age is', age)  
 print('adult')  
 else:  
 print('your age is', age)  
 print('teenager')

注意不要少写了冒号，当然上面的判断是很粗略的，完全可以用elif做更细致的判断：

age = 3  
 if age >= 18:  
 print('adult')  
 elif age >= 6:  
 print('teenager')  
 else:  
 print('kid')

if语句执行有个特点，它是从上往下判断，如果在某个判断上是True，把该判断对应的语句执行后，就忽略掉剩下的elif和else。

## 循环

（1）for循环

Python的循环有两种，一种是for...in循环，依次把list或tuple中的每个元素迭代出来，例如：

names = ['Michael', 'Bob', 'Tracy']  
 for name in names:   
 print(name)

（2）while循环

第二种循环是while循环，只要条件满足，就不断循环，条件不满足时退出循环，例如计算1—10整数之和：

sum=0  
 i=1  
 while i<=10:  
 sum=sum+i  
 i+=1  
 print(sum)

1. break结束循环

在循环中，break语句可以提前退出循环，例如：

sum=0  
 i=1  
 while i<=10:  
 sum=sum+i  
 i+=1  
 if i==5:  
 break  
 print(sum)

结果为1+2+3+4。

（4）continue进入下一个循环

在循环过程中，也可以通过continue语句，跳过当前的这次循环，直接开始下一次循环，例如：

n=0  
 while n<10:  
 n+=1  
 if n%2==0:  
 continue  
 print(n)

结果输出1，3，5，7，9。

# 函数

## Python常用内置函数

（1）数学相关

abs(a) : 求取绝对值，abs(-1)=1。

max(list) : 求取list最大值，max([1,2,3])=3。

min(list) : 求取list最小值，min([1,2,3])=1

sum(list) : 求取list元素的和，sum([1,2,3])=6。

sorted(list) : 排序，返回排序后的list，sorted([5,2,7,9,1])=[1,2,5,7,9]。

len(list/str) : 获取list/str长度，len([1,2,3])=3，len(‘中国qwe’)=5。

divmod(a,b): 获取商和余数，divmod(10,3)=(2,1)。

pow(a,b) : 获取乘方数，pow(2,4)=16。

round(a,b) : 获取指定位数的小数（四舍五入），a代表浮点数，b代表要保留的位数，round(3.145678,2)=3.15。

range(a,b,i) : 生成一个a到b，差值为i左闭右开的数组，range(2,10,2)=[2,4,6,8]。

（2）类型转换

int(str) : 转换为int型，int('1')=1，int(1.9)=1。

float(int/str) : 将int型或字符型转换为浮点型，float(1)=1.0，float(‘5.1’)=5.1。

str(int) : 转换为字符，str(1)='1'。

list(iterable) : 转换为list，list((1,2,3))=[1,2,3]，list({1,2,3})=[1,2,3]。

dict(iterable) : 转换为dict，dict([('a', 1), ('b', 2), ('c', 3)])={'a':1, 'b':2, 'c':3}。

tuple(iterable) : 转换为tuple，tuple([1,2,3])=(1,2,3)。

set(iterable) : 转换为set，set([1,4,2,4,3,5])={1,2,3,4,5}。

（3）功能相关

eval(expression, globals=None, locals=None)：执行一个表达式，或字符串作为运算，也可以把字符串转换为列表、元组、字典、集合等，只能执行单个运算表达式，不能是复杂的代码逻辑。expression：必选参数，可以是字符串。globals：可选参数，表示全局命名空间（存放全局变量），如果被提供，则必须是一个字典对象。locals：可选参数，表示当前局部命名空间（存放局部变量），如果被提供，可以是任何映射对象。如果该参数被忽略，那么它将会取与globals相同的值。用法例如：eval(“’a’+’b’”)=ab，eval(‘1+2’)=3，eval(“[‘a’,’b’]”)=[‘a’,’b’]。

exec(expression, globals=None, locals=None)：动态执行Python代码，也就是说exec可以执行复杂的Python代码，而不像eval函数那样只能计算一个表达式的值，而且与eval相比，exec()函数返回值永远都是None。

zip(iterable,...)打包函数：函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同。

str.split(char,num)拆分函数：通过指定分隔符对字符串进行切片并返回一个数组，如果参数num有指定值，则分隔num+1个子字符串。char：分隔符，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)。num：分割次数，默认为 -1, 即分隔所有。

str.strip(chars)除头去尾函数：移除字符串头尾指定的字符（chars为空时则剔除所有空字符、换行符、制表符）或字符序列，该函数表示从头或者从尾部开始进行扫描，如果扫描的字符在序列字符串中，则剔除掉，一直到遇到第一个不在序列字符串中的字符为止。同时还与str.lstrip(chars)去除头部函数，str.rstrip(chars)去除尾部函数。

str.format():字符串格式化函数，基本语法是通过{}和:来代替以前的%，可以格式化字符串也可以格式化数值。

## 函数定义

在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用return语句返回。例如自定义一个输出数值绝对值的函数：

def test1(x):  
 if x>=0:  
 return x  
 else:  
 return -x

函数体内部的语句在执行时，一旦执行到return时，函数就执行完毕，并将结果返回。如果没有return语句，函数执行完毕后也会返回结果，只是结果为None，return None可以简写为return。

**（1）空函数**

如果想定义一个什么事也不做的空函数，可以用pass语句。pass语句什么都不做，那有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass。

**（2）函数返回值**

函数可以有多个返回值，返回值其实是一个tuple元组。例如：

def test2(x,y):  
 return x+y,x\*y

test(4,5)的结果为：(9,20)。

注意：调用函数时必须要在函数后面加上()。

## 函数的参数

**（1）必选参数**

必选参数是函数中最常用的参数类型，在调用含有必选参数函数时，参数必须传入，且一个参数有且对应一个值，例如：

def test(x,y):  
 return x\*y

x与y就是必选参数。

**（2）默认参数**

定义函数参数值可以给默认值，在调用这种类型的函数，有默认值的参数可以不用给，但要注意默认值参数的位置都是在没默认值参数的后面。例如：

def test2(x,y,z=10):  
 print(x+y+z)

test(1,2)=13，test(1,2,3)=6。

**（3）可变参数**

在Python函数中，还可以定义可变参数。顾名思义，可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。可变参数定义时是在参数前面加\*，当调用函数时，函数接受到的参数是一个tuple。例如：

def test(\*number):  
 sum=0  
 for i in number:  
 sum=sum+pow(i,2)  
 print(sum)

test(1,2,3,4)=30，test(1)=1。可变参数函数可以使用列表元组集合作为参数，使用时只需要在列表元组集合前也加\*即可。例如：

list=[1,2,3,4]，test(\*list)=30。

**（4）关键字参数**

关键字参数也允许传入任意个参数，这些参数在函数内部会自动组装为一个dict字典，定义时只需要在参数名称前加\*\*。例如：

def test(\*\*number):  
 sum=0  
 for num in number:  
 sum=sum+number[num]  
 print(sum)

test(a=1,b=2,c=3)=6。关键字参数函数可以使用字典作为参数，使用时只需要在字典前也加\*\*即可。例如：

dict={‘a’:1,’b’:3,’c’:5}，test(\*\*dict)=9

**（5）设置关键字参数的关键字**

如果要限制关键字参数的名字，就可以用命名关键字参数。例如，只接收c和d作为关键字参数，这种方式定义的函数如下：

def test(a,b,\*,c,d):  
 print(a,b,c,d)

如果函数定义中已经有了一个可变参数，后面跟着的命名关键字参数就不再需要一个特殊分隔符\*，例如：

def test(a,b,\*numbers,d,e):  
 sum=a+b  
 for num in numbers:  
 sum=sum+num  
 sum=sum+d+e  
 print(sum)

这种情况下，关键字参数一定要传入，否则将会报错:test(1,2,3,4,5,6,d=7,e=8)=36

**（6）组合参数**

定义函数，可以用必选参数、默认参数、可变参数、关键字参数和命名关键字参数，这5种参数都可以组合使用，但需要注意参数定义的顺序必须是：必选参数、默认参数、可变参数、命名关键字参数、关键字参数。

## 递归函数

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，那么这个函数就是递归函数。例如计算1\*2\*3\*4......\*n：

def test(n):  
 if n==1:  
 return 1  
 return n\*test(n-1)

使用递归函数需要注意防止栈溢出，解决递归调用栈溢出的方法是通过尾递归优化， 尾递归是指在函数返回的时候，调用自身本身，并且return语句不能包含表达式。例如上述函数可优化为：

def test(n,result):  
 if n==1:  
 return result  
 return test(n-1,n\*result)

n-1,n\*result在调用函数前就会被计算，不会影响到函数。

## 装饰器

装饰器是python一个重要组成部分，它可以让其他函数或类在不需要做任何代码修改的前提下增加额外功能，装饰器是用@来修饰。看一个简单的例子：

def zhuangshi(func):  
 print('增加其他功能！')  
 return func  
@zhuangshi  
def foo(name):  
 print('my nane is',name)  
foo('boy')

这里定义了一个装饰器zhuangshi，在调用foo()函数时，实际上是先将foo作为参数传给函数zhuangshi，也就是执行zhuangshi(foo)：

def zhuangshi(foo):  
 print('增加其他功能！')  
 return foo

这里将会执行print语句，并且返回foo。而函数后面的()是执行函数，即执行返回的foo函数，并且执行函数时参数为boy。

# 高阶函数

函数可以赋值给变量，例如：

function=abs  
 function(-10)

结果为10。

函数的参数可以接受变量，例如：

b=10  
 def test(x,y):  
 return x\*y  
 test(10,b)

结果为100。

当函数的变量为函数时，这种函数就称之为高阶函数，例如：

f=pow  
 def test(x,y,function):  
 return function(x,y)  
 test(2,4,f)

结果为16。

## map()函数

map(function,Iterable)函数接收两个参数，一个是函数，另一个是可以进行迭代的对象，返回的是以迭代对象作为参数计算的函数结果并封装成一个Iterator迭代器，迭代器是惰性序列，可以通过list()或者tuple()函数返回一个list或者tuple，例如：

def test(x):  
 return pow(x,2)  
 print(list(map(test,range(5))))

结果为[0, 1, 4, 9, 16]。

## reduce()函数

reduce()函数会对参数序列中元素进行累积，具体的积累方式是将一个数据集合中的数据进行下列操作：用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，以此类推最后得到一个结果，例如：

from functools import reduce  
 def test(x,y):  
 return x\*y  
 print(reduce(test,range(1,5)))

结果为1\*2\*3\*4=24。

## filter()筛选函数

filter(function,Iterable)筛选函数，函数用于过滤序列，过滤掉不符合条件的元素。同样的也是需要两个参数，第一个为函数，第二个为序列。序列的每个元素作为参数传递给函数进行判，然后返回 True 或 False，最后将返回 True 的元素放到新列表中，例如：

def test(x):  
 return x%2==1  
 print(list(filter(test,range(1,6))))

结果为[1,3,5]。

print(list(filter(lambda x:x%2==0,range(1,6))))

结果为[2,4]。

## sorted()排序函数

sorted(Iterable, key=None, reverse=False)排序函数，用来排序可迭代的对象，结果返回一个list。Key：主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。reverse：排序规则，reverse = True 降序，reverse = False 升序（默认）。对一个列表进行排序，例如：

list=[9,2,6,5,1]  
 print(sorted(list))

也可以对字典的key、value、item进行排序，例如：

dict={'a':9,'h':2,'c':7,'e':5,'g':10}  
 print(sorted(dict.keys(),reverse=True))  
 print(sorted(dict.values(),reverse=False))  
 print(sorted(dict.items(),key=lambda x:x[1],reverse=False))  
 print(sorted(dict.items(),key=lambda x:x[0],reverse=False))

结果依次为：['h', 'g', 'e', 'c', 'a']、[2, 5, 7, 9, 10]、[('h', 2), ('e', 5), ('c', 7), ('a', 9), ('g', 10)]、[('a', 9), ('c', 7), ('e', 5), ('g', 10), ('h', 2)]。

list自带的函数sort()与sorted()函数很像，但是sort()只能应用在list，sorted()可以对所有可迭代的对象进行排序操作。sort()是对已经存在的列表进行操作，无返回值，而sorted()方法返回的是一个新的list，不会对传入的list产生影响。例如：

List=[10,3,5,7,1,12]  
 print(sorted(List))  
 print(List)  
 List.sort()  
 print(List)

结果为：[1, 3, 5, 7, 10, 12]、[10, 3, 5, 7, 1, 12]、[1, 3, 5, 7, 10, 12]，List使用sorted()排序后不会影响到List本身，而List使用自带的sort()则List会进行排序。

# 高级特性

## 切片

切片根据索引可以取出列表元组指定范围的数，例如元组tuple=(‘a’,’b’,’c’,’d’,’e’)，取出这个元组的第1个到第3个数写法为tuple[0:3]，其中索引3不包括，结果为(‘a’,’b’,’c’)。如果想要从第n个值取到最后一个值，则这么写tuple[n:]。切片也支持用索引取出倒数的数，例如tuple[-3:]取最后3个数，结果是(‘c’,’d’,’e’)。切片还有其他的操作，例如每多少个中取一个：

liebiao=list(range(100))，liebiao[:10:2]：前十个每两个取一个，liebiao[::5]：每五个取一个。此外切片还可以对字符串进行操作，例如str=’abcdefg’，str[:4:3]=’ad’，str[::2]=’ac’。

## 迭代

如果给定一个list或tuple，可以通过for循环来遍历这个list或tuple，这种遍历称为迭代，例如:

List=list(range(0,10,2))  
 for i in List:  
 print(i)

结果为：0,2,4,6,8。

Python可以进行迭代的对象不仅局限于list或tuple，dict、set、str也可以进行迭代，dict迭代的结果是字典的key值。判断一个对象是否可以进行迭代，可以通过collections.abc模块的Iterable类型：

from collections.abc import Iterable  
 print(isinstance(['a','b','c'],Iterable))  
 print(isinstance(('a','b','c'),Iterable))  
 print(isinstance('abc',Iterable))

结果为True,True,True。

enumerate(Iterable,start=开始下标) 函数用于将一个可遍历的数据对象组合为一个索引序列，同时列出数据和数据下标。需要注意的是开始下标并不是迭代数据的开始时游标，只是起到一个开始编号的作用，例如：

tuple=('a',1,'b',2)  
 for i,value in enumerate(tuple,start=2):  
 print(i,':',value)

结果为2:a,3:1,4:b,5:2。

## 列表生成式

生成一个list，常用的方法是利用list函数进行转换，例如：

List=list(range(10))  
 Set={1,2,3,4}  
 Tuple=(1,2,3,4)  
 list(Set)  
 list(Tuple)

也可以通过for循环来创建列表，例如：

[ x for x in range(10)]

结果为[0,1,2,3,4,5,6,7,8,9]。

for循环可以使用多层，例如：

[m+n for m in range(5) for n in range(5,10)]

结果为[5,7,9,11,13]。

在for循环中还可以使用判断语句，例如：

[pow(x,2) for x in range(10) if x%2==0]

结果为[0,4,16,36,64]。

如果想要在for循环中使用if......else，if......else必须写在for前面，例如：

[x if x>=0 else -x for x in range(-4,5)]

结果为[4,3,2,1,0,1,2,3,4]。

## lambda()匿名函数

lambda函数（匿名函数）：lambda argument\_list:expression。argument\_list是参数列表，它的结构与Python中函数(function)的参数列表是一样的（输入）。expression是一个关于参数的表达式，表达式中出现的参数需要在argument\_list中有定义，并且表达式只能是单行的（输出）。例如：

function=lambda x,y:x\*y  
 function(4,5)

结果为20。

# 模块

在计算机程序的开发过程中，随着程序代码越写越多，在一个文件里代码就会越来越长，越来越不容易维护。为了编写维护性比较的代码，一般会对函数进行分组，分别放到不同的文件里，这样，每个文件包含的代码就相对较少，很多编程语言都采用这种组织代码的方式。在Python中，一个.py文件就称之为一个模块。

使用模块最大的好处是大大提高了代码的可维护性，其次，编写代码不必从零开始。当一个模块编写完毕，就可以被其他地方引用。在编写程序的时候，也经常引用其他模块，包括Python内置的模块和来自第三方的模块。使用模块还可以避免函数名和变量名冲突，相同名字的函数和变量完全可以分别存在不同的模块中。因此，在编写模块时，不必考虑名字会与其他模块冲突。但是也要注意，尽量不要与内置函数名字冲突。

如果不同的人编写的模块名相同怎么办？为了避免模块名冲突，Python又引入了按目录来组织模块的方法，称为包（Package）。举个例子，比如有两个abc.py文件，这个时候可以通过包来组织模块。方法是选择两个包，比如mycompany1、mycompany2，然后把这两个文件分别存放到这两个包的目录下。那这时候，两个模块的名称就变成了mycompany1.abc、mycompany2.abc。需要注意的是每一个包目录下面都会有一个\_\_init\_\_.py的文件，这个文件是必须存在的，否则，Python就把这个目录当成普通目录，而不是一个包。\_\_init\_\_.py可以是空文件，也可以有Python代码，因为\_\_init\_\_.py本身就是一个模块，而它的模块名就是mycompany。类似的，可以有多级目录，组成多级层次的包结构。创建模块时要注意命名，不能和Python自带的模块名称冲突。例如，系统自带了sys模块，创建的模块就不可命名为sys.py，否则将无法导入系统自带的sys模块。

## 创建模块

创建模块其实就是建一个.py文件，通常情况下会把类似的功能代码写在同一个.py文件中，例如以下test.py文件就是一个模块：

# -\*- coding: utf-8 -\*-  
 class Moudle():  
 def func1(x,y):  
 return x\*y-(x+y)  
 def func2(x,y):  
 return pow(x,y)-(x+y)  
 if \_\_name\_\_ == '\_\_main\_\_':  
 print(Moudle.func1(4,2))  
 print(Moudle.func2(4,2))

首行信息指定了运行该文件代码使用的编码格式，该模块中有一个类，类有两个函数。

## 使用模块

使用模块前，需要导入模块，导入的时候需要注意如果引用的模块在不在包里面，那么导入不能使用相对路径的导入（例如：from ..test import Test），否则会报找不到父包的错误信息。同时需要注意，如果导入的模块中有执行语句，那么一定要将执行语句写在if \_\_name\_\_ == '\_\_main\_\_':判断语句里面，否则引用这个模块时，去执行将会执行引用模块中的执行语句。例如，上述创建的模块如果没有把执行语句写在if \_\_name\_\_ == '\_\_main\_\_'，那么再其他地方引用该模块时，将会运行这两条执行语句。（\_\_name\_\_是Python的内建变量，每个.py文件都有这个变量，该变量可以显示模块是被自己执行的还是其他模块调用的。如果是自己执行的，把么\_\_name\_\_就等于\_\_main\_\_，如果是被别的模块调用，则显示模块的绝对路径信息）。

## 安装第三方模块

Python的第三方模块一般会集成到包中，使用第三方模块需要下载对应的包。Pip是Python包管理工具，Pip提供了下载包、卸载包以及升级包等操作。使用Pip前也需要进行安装，但Pip从Python3.4开始就默认包含在Python安装程序中，安装了Python就包含了pip。

Windows系统中可以直接在doc命令下操作Pip，Pip有以下常用的操作命令：

（1）pip install packagename

下载并安装包，packagename为包的名称，如果想要下载指定版本的包，则需要在包后面加上版本的信息，例如pip install xlwt==1.2.0。如果不加上版本信息，那下载安装的包将是最新版本的。如果需要更新包，那么命令为pip install --upgrade packagename。

（2）pip download packagename

下载包但不安装，用法与install类似。

（3）pip uninstall packagename

卸载包。

（4）pip list/pip freeze

列出已经安装的包以及版本信息，使用pip list --outdated可以列出可以更新的包，包括当前安装的版本信息以及最新版本信息。

# 面向对象编程

面向对象编程是一种程序设计思想，在程序设计中被广泛运用。当要完成一件事情的时候，面向对象会把事情涉及到的事物抽象成对象，赋予对象一些属性和方法，然后用对象的方法去完成事情。例如用家里的洗衣机洗衣服这件事，涉及到的对象洗衣机。洗衣机的品牌可能有海尔、美的，洗衣的颜色有白色或者灰色，像这些对于洗衣机的静态描述，设计成程序语言时那就是对象的属性，是用来描述对象的特征。洗衣机还有其功能，比如洗涤、漂洗、脱水、烘干，这些在设计程序时就是对象的方法，描述对象能做什么事情。

## 类和实例

面向对象最重要的概念就是类（Class）和实例（Instance），类是抽象的模板，用来描述具有相同属性和方法的对象的集合，实例是根据类创建出来的一个个具体的对象，每个对象都拥有相同的属性与方法，但各自的数据可能不同。

在Python中，定义类通过class关键字，关键字后面紧跟着是类的名称，类名通常是大写字母开头的单词，紧接着是(父类列表)，表示该类从哪些类继承下来。如果没有合适的继承类，括号里面可以空着或者省略括号，也可以使用object类，这是所有类最终都会继承的类，括号内空着跟写object是一样的，基本结构如下：

class Classname(object):  
 def test(self,x,y):  
 print(x\*y)

通过调用类的实例化方法来创建一个实例（即对象）：instance=Classsname()，实例拥有类一样的属性及方法，可以直接调用类的方法：instance.test(8,9)=72，实例化时别忘了类名后面的()。

类中的方法与普通的方法有一个很不一样的地方，就是类中的方法第一个参数是self，且在调用类中的函数时self是不需要传入的。这个self其实代表的是实例化对象的本身，实例化对象调用函数时会自动传入这个参数，这样可以清楚知道是哪个对象调用这个方法。self还有一个非常大的用途就是可以让属性全局化，例如：

from selenium import webdriver  
import unittest  
class Test(unittest.TestCase):  
 # 通过self.browser将webdriver.Ie()变成全局属性绑定在实例化对象上，其他方法既可以直接进行使用。  
 def setUp(self):  
 self.browser=webdriver.Ie()  
 browser=self.browser  
 browser.maximize\_window()  
 browser.implicitly\_wait(5)  
 browser.get('https://www.baidu.com')  
 def test1(self):  
 browser=self.browser  
 browser.find\_element\_by\_id('kw').send\_keys('python')  
 browser.find\_element\_by\_id('su').click()  
 def test2(self):  
 browser = self.browser  
 browser.find\_element\_by\_id('kw').send\_keys('自动化测试')  
 browser.find\_element\_by\_id('su').click()  
 def tearDown(self):  
 browser=self.browser  
 browser.quit()  
if \_\_name\_\_=='\_\_main\_\_':  
 unittest.main()

## 继承和多态

在定义一个新的类时，可以从某个现有的类继承，这种类称之为子类，被继承的类称之为基类、父类或者超类，继承最大的好处就是子类获取了被继承类的全部功能，例如：

class Jicheng(Classname):  
 pass

Jicheng类继承了Classname，可以直接使用Classname的方法，例如Jicheng().test(5,6)=30。继承还可以一级一级地继承下来，就好比从爷爷到爸爸、再到儿子这样的关系。而任何类，最终都可以追溯到根类object，这些继承关系看上去就像一颗倒着的树。可以使用Python的issubclass(child,father)函数来检查继承类，返回的结果是布尔类型。

当子类和父类都存在相同的方法时，子类的方法会覆盖了父类的，在代码运行时，会调用子类的方法，这就是类的多态。例如：

class Jicheng(Classname):  
 def test(self,x,y):  
 print(x,'乘以',y,'等于',x\*y)

Jicheng().test(5,6)结果为：x乘以y等于30。类可以多重继承，继承多个类，类的多重继承使子类的功能更加丰富。

## 获取对象信息

对象有属性、方法等信息，可以通过函数来获取对象的这些信息。

1. type(object)：查询对象的基本类型。
2. isinstance(object,(type1,type2))：判断对象是否属于哪种类型，返回的是布尔型。
3. dir(object)：查询对象的所有属性及方法。
4. hasattr(object, name)：判断对象中是否含有name属性，返回的是布尔型，name的参数是string类型。
5. getattr(object,name,defaultvalue)：返回对象属性的值，如果对象不包含该属性，则返回defaultvalue。如果name是函数名，则返回是函数本身。
6. setattr(object,name,value)：给对象的name属性赋值，如果属性name不存在，则会给对象增加属性，如果存在则改变对象的属性。

## 绑定属性及方法

类和实例可以绑定新的属性及方法，如果绑定到类上，那么通过这个类创建的实例都能进行访问。绑定在实例上，那么绑定的属性及方法是类独有的，其他实例不能进行访问。例如：

class Test():  
 pass

Test.name='类的名称'  
 Instance1=Test()  
 Instance2=Test()

Instance2.name='实例2的名称'

print(Test.name)  
 print(Instance1.name)  
 print(Instance2.name)

结果为：类的名称、类的名称、实例2的名称。

给实例添加方法需要从types中导入MethodType：

from types import MethodType  
 class Test():  
 pass  
 def mult(self,x,y):  
 return x\*y  
 def sum(self,x,y):  
 return x+y  
 Test.mult=mult  
 Instance1=Test()  
 Instance2=Test()  
 Instance2.sum=MethodType(sum,Instance2)  
 print(Instance1.mult(3,4))  
 print(Instance2.sum(1,99))  
 print(Instance2.mult(4,5))

结果为：12，100，20。

给类添加方法直接classname.method=method即可，不需要使用MethonType()来添加，类添加的方法与属性实例也可以进行访问。注意实例添加的属性及方法名称不能与类的属性及方法名称重复，否则实例在访问时只能访问到实例本身的属性及方法，而不能访问类的属性及方法。如果需要删除对象的方法及属性，可以使用del object.method/object.attribute进行删除。

想要限制实例的属性怎么办？比如，只允许对实例添加name和age属性。为了达到限制的目的，Python允许在定义class的时候，定义一个特殊的\_\_slots\_\_变量，来限制该class实例能添加的属性：

class Test():  
 \_\_slots\_\_=('name','age')  
 pass  
 instance=Test()

那么instance只能允许添加name和age属性，\_\_slots\_\_定义的属性仅对当前类实例起作用，对继承的子类不起作用，例如：

class Child(Test):  
 pass  
 childinstance=Child()

childinstance实例就可以添加name和age属性之外的属性。

## 类的初始化函数

python有许多\_\_\*\_\_两个下划线开头的函数，两个下划线表示该方法为私有，不能在类的外部被使用或访问。\_\_init\_\_函数是一种经常在python类中使用的私有方法，当类实例化一个对象时，\_\_init\_\_会直接调用，它一般是用来做初始化工作的。如果\_\_init\_\_函数中包含除self以外的参数，那么在实例化时，需要要传入，否则将会报错。

class Test:  
 # 类的初始化函数  
 def \_\_init\_\_(self,x,y,z):  
 self.x=x  
 self.y=y  
 self.z=z  
 def sum(self):  
 return self.x+self.y+self.z  
 def mix(self):  
 return self.x\*self.y\*self.z  
# 如果类的初始换函数含有除了self之外的参数，那么类的实例化需要传入这些参数。  
test=Test(2,3,4)  
print(test.sum())  
print(test.mix())

# 错误处理

## 错误处理机制try

在程序运行的过程中，如果发生了错误，可以事先约定返回一个错误信息，这样可以帮助快速找到错误的地方以及错误的原因。高级语言通常都内置了一套try...except...finally...的错误处理机制，Python也不例外。

try模块来执行语句，如果遇到错误，try模块错误后面的代码将不会执行（注意：try模块中的语句执行发生异常将不会影响程序的执行，而不在try模块中执行的语句发生异常则在发生异常时，语句会停止执行），然后直接跳转至错误处理代码：except。except如果捕捉到对应的错误（except可以写多个，且如果多个都能匹配到错误原因只会执行一个，且是最上面的那个，错误信息也可以输出），则将会执行except模块，执行完except模块，如果有finally（也可以没有finally），则执行finally语句块，至此执行完毕。finally最后执行，且一定会执行。例如：

try:  
 print('执行10除以0')  
 print(10/0)  
 print('出现了错误，后面的将不会执行')  
 print('执行10除以字符串5')  
 print(10/'5')  
 except TypeError as E:  
 print('被除数只能是数值')  
 print(E)  
 except ZeroDivisionError as E:  
 print('捕捉到相应的错误，我将会执行')  
 print('0不能作为除数！')  
 print(E)  
 finally:  
 print('不管try模块是否发生错误，我都会执行，且最后执行！')

Python的错误类型有很多，相对应判断的方法也很多，可以使用通用异常Exception:

try:  
 print('执行10除以字符串5')  
 print(10/'5')  
 print('执行10除以0')  
 print(10/0)  
 except Exception as E:  
 print(E)

可以在except语句块后面加一个else，如果没有错误发生时，会执行else语句：

try:  
 print('执行10除以2')  
 print(10/2)  
 print('执行5乘以2')  
 print(5\*2)  
 except Exception as E:  
 print(E)  
 else:  
 print('try模块中没有发生错误，将会执行else模块。')

使用try...except捕获错误还有一个好处是可以跨越多层调用，比如函数main()调用foo()，foo()调用bar()，结果bar()出错了，这时，只要main()捕获到了，就可以处理：

def bar(x):  
 return 10/x  
 def foo(x):  
 return bar(x)\*2  
 def main(x):  
 try:  
 return foo(x)+2  
 except Exception as E:  
 print(E)  
 finally:  
 print('我最后执行')

如果执行函数main(0)，则函数bar(x)将会报错，则也会处理。

## 错误信息栈

执行多层语句、嵌套语句或者引用模块时出现错误，那么错误会就会一直往上抛，直到出现错误最根本的地方，打印出一个错误信息。例如：

def bar(x):  
 return 10/x  
 def foo(x):  
 return bar(x)\*2  
 def main(x):  
 foo(x)+2

执行mian(0)时，将会抛出：

Traceback (most recent call last):

File "E:/Study-Python/Python基础/调试/处理错误信息.py", line 8, in <module>

main(0)

File "E:/Study-Python/Python基础/调试/处理错误信息.py", line 6, in main

foo(x)+2

File "E:/Study-Python/Python基础/调试/处理错误信息.py", line 4, in foo

return bar(x)\*2

File "E:/Study-Python/Python基础/调试/处理错误信息.py", line 2, in bar

return 10/x

ZeroDivisionError: division by zero

解读错误信息是定位错误的关键，从上往下可以看到整个错误的调用函数链：

错误信息第1行：Traceback (most recent call last)，回溯（最近一次调用）

错误信息第2行：表示mian(0)模块执行错误。

错误信息第3行：表示mian模块执行错误是执行了foo(x)+2错误。

错误信息第4行：表示foo模块执行错误是执行了bar(x)\*2错误。

错误信息第5行：表示bar模块执行错误是执行了 10/x错误。

那么错误的原因就是执行10/0时发生了错误，错误信息ZeroDivisionError: division by zero也表示了0作为除数时是错误的。

## 自定义错误处理

执行Python语句时如果出现系统上错误，Python会自动引发异常。但如果不是系统上的异常，比如输入信息不符合要求、输出结果与期望不符等错误，可以通过raise语句来自定义抛出错误信息，抛出的错误信息可以被捕获。当raise抛出异常，后面的语句将不能执行。

def Diomod(x,y):  
 try:  
 if not(isinstance(x,(int,float)) and isinstance(y,(int,float))):  
 raise TypeError('被除数或除数的类型不对！')  
 if y==0:  
 raise ValueError('除数不能为0！')  
 except Exception as E:  
 print('输入错误：',E)  
 else:  
 print('输入正确，结果为：',x/y)  
 if \_\_name\_\_ == '\_\_main\_\_':  
 Diomod(10,0)  
 Diomod('a',1)  
 Diomod(10,2)

结果为：（1）输入错误： 除数不能为0！、（2）输入错误： 被除数或除数的类型不对！、（3）输入正确，结果为： 5.0。

## 断言assert

可以通过assert断言的方式来检查错误，assert的基本语法为：assert 表达式,”断言语句”。如果表达式返回的结果是False时，则抛出AssertionError并抛出断言语句，断言异常可以被except捕获：

def Diomod(x,y):  
 try:  
 assert isinstance(x,(int,float)) and isinstance(y,(int,float)),'被除数与除数类型不正确'  
 assert y!=0,'除数不能为0'  
 except Exception as E:  
 print('输入错误：',E)  
 else:  
 print('输入正确，结果为：',x/y)  
 if \_\_name\_\_ == '\_\_main\_\_':  
 Diomod(10,0)  
 Diomod('a',1)  
 Diomod(10,2)

结果为：（1）输入错误： 除数不能为0！、（2）输入错误： 被除数或除数的类型不对！、（3）输入正确，结果为： 5.0。

## logging模块记录日志

logging模块是Python内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志信息内容等。使用logging模块需要先导入import logging，然后使用logging.basicConfig(filename=’’,filemode=’’,level=’’,datefmt=’’,forMat=’’)函数进行配置。

filename：日志存放的文件，例如filename=r'D:\Python\test.log'；filemode：记录日志信息的方式，方式有a与w，a是追加信息，w会先清空日志信息然后再写入新的信息；level：记录信息的等级，从低到高为DEBUG ,INFO ,WARNING ,ERROR, CRITICAL。对应写入日志信息的方法为logging.debug(message)、logging.info(message)、logging.warning(message)、logging.er

ror(message)、logging.critical(message)。level如果不设置，那么默认值为WARNING，日志级别大于等于WARNING的才会被输出；datefmt：指定日期时间格式；forMat：日志信息内容，常用的参数有：%(asctime)s字符串形式的当前时间（默认格式是20200320-01-01 12:12:12,896）、%(filename)s调用日志输出函数的模块的文件名、%(lineno)d调用日志输出函数的语句所在的代码行、%(levelname)s文本形式的日志级别、%(message)s用户输出的消息等。

import logging  
 def Diomod(x,y):  
 try:  
 if not(isinstance(x,(int,float)) and isinstance(y,(int,float))):  
 logging.debug('被除数或除数的类型不对！')  
 raise TypeError('被除数或除数的类型不对！')  
 if y==0:  
 logging.info('除数不能为0！')  
 raise ValueError('除数不能为0！')  
 except Exception as E:  
 print('输入错误：',E)  
 else:  
 print('输入正确，结果为：',x/y)  
 if \_\_name\_\_ == '\_\_main\_\_':  
 logging.basicConfig(filename=r'D:\Python\Log.log', filemode='a', level=logging.DEBUG,datefmt='%Y-%m-%d %H:%M:%S', format="%(asctime)s %(filename)s[line:%(lineno)d]%(levelname)s-%(message)s")  
 Diomod(10,0)  
 Diomod('a',1)  
 Diomod(10,2)  
 logging.warning('warning message!')  
 logging.error('error message!')  
 logging.critical('critical message!')

运行以上代码，输出的日志为：

2020-03-31 16:45:31 logging模块记录日志.py[line:9]INFO-除数不能为0！

2020-03-31 16:45:31 logging模块记录日志.py[line:6]DEBUG-被除数或除数的类型不对！

2020-03-31 16:45:31 logging模块记录日志.py[line:22]WARNING-warning message!

2020-03-31 16:45:31 logging模块记录日志.py[line:23]ERROR-error message!

2020-03-31 16:45:31 logging模块记录日志.py[line:24]CRITICAL-critical message!

## 单元测试

单元测试是用来对一个模块、一个函数或者一个类来进行正确性检验的测试工作。比如对函数Test，可以编写出以下几个测试用例：

def Test(x):  
 if not isinstance(x,(int,float)):  
 return '输入参数类型不正确，需要输入数值！'  
 elif x>0:  
 return x  
 else:  
 return -x  
 if \_\_name\_\_ == '\_\_main\_\_':  
 print(Test('10'))  
 print(Test('A'))  
 print(Test('%'))  
 print(Test(10))  
 print(Test(-5))  
 print(Test(0))

如果单元测试通过，说明测试的这个函数能够正常工作。如果单元测试不通过，要么函数有bug，要么测试条件输入不正确。

# 数据读写操作

## 读写文本文件

使用Python的open()函数可以打开文件，open()函数有八个参数，常用的是三个参数fileName(文件路径名称)、mode(打开方式，常用的值有r、r+、w、w+)、encoding(打开文件的编码方式，默认是utf-8)。mode常用的方式有r(只读文件)、r+(读写方式打开)、w(只写方式打开，如果文件存在则打开已有文件，且会删除原先的内容，如果不存在则创建文件)、w+(以读写方式打开文件，参照w)，r必须要有对应的文件，或者会抛出IOError，w可以没有。

**（1）读文本文件**

打开文件其实就是打开了文件对象，用open()方法打开文件可以实例化对象，然后调用对象的方法。打开文件之后，然后调用文件对象的read()方法读取文件内容，读取完文件内容后，还需要调用文件对象的close()方法关闭文件：

file=open(r'D:\Python\read.txt','r')  
 print(file.read())  
 file.close

read()方法会一次性读取文件的所有内容，如果文件比较大，那么最好指定一下read(size)的size(字节内容)，也可以使用readline()方法每次读取一行内容：

file=open(r'D:\Python\read.txt','r')  
 print(file.readline())  
 print(file.readline())  
 print(file.readline())  
 print(file.readline())  
 file.close

readlines()方法可以读取所有数据，并返回以行为元素的列表，

file=open(r'D:\Python\read.txt','r')  
 for line in file.readlines():  
 print(line.strip())  
 file.close()

每次读取文件，最后还需要关闭文件，这种写法比较繁琐，可以用Python的with语句来写：with open(filename,’r’)，这样子可以省略close。

with open(r'D:\Python\read.txt','r') as file:  
 print(file.read())

**（2）写文本文件**

往文本文件中写入内容的写法与读差不多，也是要先open()文件对象，然后在调用对象的写入方法，写入内容后，要关闭文件。

file=open(r'D:\Python\write.txt','w',encoding='utf-8')  
 file.write('第一行')  
 file.close()

需要换行写入内容时，可以在前一行内容中加入换行符，或者是新的内容之前加入换行符。

file=open(r'D:\Python\write.txt','w',encoding='utf-8')  
 file.write('第一行')  
 file.write('\n第二行\n')  
 file.write('第三行\n')  
 file.write('第四行')  
 file.close()

写入数据同样也可以使用with的写法。

with open(r'D:\Python\write.txt','w',encoding='utf-8') as file:  
 file.write('第一行')

## 读写excel文件

1. **读取excel数据**

读取Excel文件中的数据需要借助第三方模块xlrd，xlrd提供了以下常用的方法对Excel文件进行操作：

Excel=xlrd.open\_workbook(filename,..)：打开一个Excel表格。

Sheet=Excel.sheets()[n]：通过sheets的索引获取sheet。

Sheet=Excel.sheet\_by\_index(n)：通过sheets的索引获取sheet。

Sheet=Excel.sheet\_by\_name(sheetname)：通过sheet的name获取sheet。

row=sheet.row\_values(n)：获取sheet第n+1行数据。

col=sheet.col\_values(n)：获取sheet第n+1列数据。

row(n).value：获取这一行中第n+1列数据。

col(n).value：获取这一列中第n+1行数据。

Sheet.cell(n,m).value：获取第n+1行，m+1列数据。

Sheet.nrows：工作表的行数。

Sheet.ncols：工作表的列数。

import xlrd  
# 实例化excel对象。  
excel=xlrd.open\_workbook(r'D:\自动化测试\data.xlsx')  
# 实例化sheet对象。  
sheet=excel.sheets()[1]  
sheet=excel.sheet\_by\_index(1)  
sheet=excel.sheet\_by\_name('Sheet1')  
# 获取sheet行数。  
print(sheet.nrows)  
# 获取sheet列数。  
print(sheet.ncols)  
# 获取sheet某行数据。  
row=sheet.row(1)  
print(row)  
# 获取sheet某列数据。  
col=sheet.col(0)  
print(col)  
# 获取某行的某列值。  
print(row[1].value)  
# 获取某列的某行值。  
print(col[0].value)  
# 获取sheet某行某列值。  
print(sheet.cell(2,0).value)

1. **excel写入数据**

往Excel文件中写入数据需要借助第三方模块xlwt或者xlsxwriter。

import xlwt  
# 创建excel对象。  
excel=xlwt.Workbook(encoding='utf-8')  
# 创建sheet工作表。  
sheet=excel.add\_sheet('工作表1')  
# 往sheet中写入数据。  
sheet.write(4,5,'中国')  
# 生成excel文件。  
excel.save(r'D:\自动化测试\xlwt.xlsx')

import xlsxwriter  
# 实例化excel对象。  
excel=xlsxwriter.Workbook(r'D:\自动化测试\xlsxwriter.xlsx')  
# 添加sheet。  
sheet=excel.add\_worksheet('工作表1')  
# 写入数据。  
sheet.write(2,3,10000)  
sheet.write\_string(3,4,'中国')  
sheet.write('F6','浙江省杭州市余杭区')  
# 关闭文件（必须执行这一步）  
excel.close()

## 操作目录

如果要操作文件、目录，可以在命令行下面输入操作系统提供的各种命令来完成。比如dir、cp等命令。如果要在Python程序中执行这些目录和文件的操作怎么办？其实操作系统提供的命令只是简单地调用了操作系统提供的接口函数，Python内置的os模块也可以直接调用操作系统提供的接口函数。

操作文件和目录的函数一部分放在os模块中，一部分放在os.path模块中，这一点要注意一下，常用的目录操作函数如下：

os.getcwd()函数：获取执行该函数的文件所在目录的绝对路径。

os.mkdir(path)函数：创建单级目录，需要保证path里父级目录必须存在且创建的目录不存在,否则将会报错。

os.rmdir(path)函数：删除单级目录，需要保证目录必须存在且为空目录，否则将会报错。

os.makedirs(path)函数：如果path里父级目录不存在，则会一起创建，需要保证创建的最底层目录不存在，否则将会报错。

os.removedirs(path)函数：如果path里有多级目录，则会删除多级目录，顺序是先删除子目录，然后再删除父目录，且目录为空时才能删除。

os.chdir(path)函数：改变当前文件的目录路径。

os.rename(oldpath,newpath)函数：重命名文件或目录。

os.path.join(path,filename)函数：路径合成函数。

os.path.split(path)函数：拆分路径信息，返回两个元素的元组，后面一个元素为末级目录或者是文件名。

os.path.splitext(path)函数：拆分路径信息，返回两个元素的元组，后面一个元素为文件扩展名。

os.path.realpath(\_\_file\_\_)函数：获取当前文件的绝对路径（包括文件名及扩展名）。

## 序列化

变量从内存中变成可存储或传输的过程称之为序列化，在Python中叫pickling，序列化之后，就可以把序列化后的内容写入磁盘，或者通过网络传输到别的机器上。反过来，把变量内容从序列化的对象重新读到内存里称之为反序列化，即unpickling。 Python提供了pickle模块来实现序列化：

import pickle  
 # 变量的序列化  
 Dict=dict(name='LiMing',age=20,addres='浙江省温州市泰顺县')  
 Pickle=pickle.dumps(Dict)  
 print(Pickle)  
 # 序列化变量写入文件  
 with open(r'D:\Python\pickle.txt','wb') as file:  
 pickle.dump(Dict,file)  
 # 变量的反序列化  
 print(pickle.loads(Pickle))  
 # 从文件中读取序列化变量数据  
 with open(r'D:\Python\pickle.txt','rb') as file:  
 Pkdict=pickle.load(file)  
 print(Pkdict)

pickle.dumps()方法把任意对象序列化成一个bytes，读写需要以二进制的方式进行。但是这种序列化方式的信息不能跨语言，且不同版本可能还不兼容。如果要在不同的编程语言之间传递对象，就必须把对象序列化为标准格式，比如XML，更好的方法是序列化为JSON，因为JSON表示出来就是一个字符串，可以被所有语言读取，也可以方便地存储到磁盘或者通过网络传输。JSON不仅是标准格式，并且比XML更快，而且可以直接在Web页面中读取，非常方便。Python内置的json模块提供了非常完善的Python对象到JSON格式的转换：

import json  
 # 变量序列化  
 Dict=dict(name='LiMing',age=20,addres='浙江省温州市泰顺县')  
 Json=json.dumps(Dict,ensure\_ascii=False)  
 print(Json)  
 # 序列化变量写入文件  
 with open(r'D:\Python\json.txt','w') as file:  
 json.dump(Dict,file,ensure\_ascii=False)  
 # 变量反序列化  
 print(json.loads(Json))  
 # 从文件中读取序列化变量数据  
 with open(r'D:\Python\json.txt','r') as file:  
 Jsdict=json.load(file)  
 print(Jsdict)

如果json内容中包含中文内容，使用json的dump方法需要传入ensure\_ascii=False，否则将会全部内容都会进行ASCII编码，不能显示中文内容。

# 进程

操作系统中一个任务就是一个进程（Process），比如打开一个浏览器就是启动一个浏览器进程，打开一个Word就启动了一个Word进程。有些进程同时会干几件事，比如Word，它可以同时进行打字、拼写检查、打印等事情。在一个进程内部，要同时干多件事，就需要同时运行多个“子任务”，进程内的这些“子任务”称为线程（Thread）。进程是操作系统的基本单位，而线程是任务调度和执行的基本单位，一个进程至少有一个线程。如果需要执行多个任务，有以下解决方案：一、启动多个进程，每个进程一个线程。二、启动一个进程，在一个进程内启动多个线程。三、启动多个进程，每个进程再启动多个线程，这种模型比较复杂，实际很少采用。

## 创建进程

Python multiprocessing模块提供了Process 类，使用该类创建实例化对象就是创建进程Process=Process(target=,name=,args=(),kwargs={})。target：为新建进程指定执行任务，一般指定一个函数或者方法。name：为新建进程设置名称，可为空。args：非关键字参数，函数的参数。kwargs：关键字参数，函数的参数。

执行Python代码时其实已经启动了一个进程负责执行代码，这个进程称之为主进程，创建的进程称之为子进程。创建子进程去执行函数最大的好处就是如果需要调用多个函数，那么可以同时并发去执行，例如：

from multiprocessing import Process  
 import datetime,random,os  
 from time import sleep  
  
 def func1(ProcessName):  
 for i in range(5):  
 sleep(random.random())  
 print(ProcessName,'(',os.getpid(),')','在执行')  
 def func2(ProcessName):  
 for i in range(5):  
 sleep(random.random())  
 print(ProcessName,'(',os.getpid(),')','在执行')  
 if \_\_name\_\_ == '\_\_main\_\_':  
 Process1=Process(target=func1,args=('进程1',),name='进程1')  
 Process2=Process(target=func2,args=('进程2',),name='进程2')  
 # 开始运行进程  
 Process1.start()  
 Process2.start()  
 Process1.join()  
 Process2.join()

运行代码时，函数1跟函数2将会同步执行，进程对象有以下几个常用的方法：

start()：启动进程。

join(timeout)：主进程会在调用该方法的进程执行完后或者设置的超时时间之后才会执行，例如以下代码：

if \_\_name\_\_ == '\_\_main\_\_':  
 Process1=Process(target=func1,args=('进程1',),name='进程1')  
 Process2=Process(target=func2,args=('进程2',),name='进程2')  
 # 开始运行进程  
 Process1.start()  
 Process2.start()  
 Process1.join()  
 Process2.join()  
 print('main')

print(‘main’)会在Process1与Process2执行完后才会执行，如果Process1、Process2没有调用join()方法，那么程序很可能会先调用print(‘main’)。如果是以下代码，则Process2会在Process1执行完后才会执行，因为执行Process2.start()是主线程，Process1调用了join()方法，那么主线程会等到Process1执行完后才会执行Process2.start()。

if \_\_name\_\_ == '\_\_main\_\_':  
 Process1=Process(target=func1,args=('进程1',),name='进程1')  
 Process2=Process(target=func2,args=('进程2',),name='进程2')  
 # 开始运行进程  
 Process1.start()

Process1.join()  
 Process2.start()  
 Process2.join()  
 print('main')

terminate()：强制终止进程。

is\_alive()：如果进程还在执行，则返回True。

name：进程的名称，可以重命名。

pid：进程的ID。

daemon：是否为守护进程，默认为False，设置时需要在进程调用start()前设置，如果设为True则为守护进程。守护进程会随着父进程的结束而结束，普通子进程运行完父进程才会结束，即使父进程代码块已经运行完毕。例如以下代码，那么子进程Process很可能还未执行其函数就在主进程执行完print(‘main’)就结束了。

if \_\_name\_\_ == '\_\_main\_\_':  
 Process=Process(target=func,args=('子进程',),name='子进程')  
 # 开始运行进程  
 Process.daemon = True  
 Process.start()  
 print('main')

## 进程池

如果要启动大量的子进程，可以用进程池的方式批量创建子进程，multprocessing模块提供的Pool可以用来创建进程池：

from multiprocessing import Pool  
 import os,datetime,random  
 from time import sleep  
 def func(n):  
 for i in range(5):  
 sleep(random.random())  
 print('第',n,'个进程(ID：',os.getpid(),')','在执行')  
 if \_\_name\_\_ == '\_\_main\_\_':  
 Process=Pool(4)  
 for n in range(1,5):  
 Process.apply\_async(func,args=(n,))  
 Process.close()  
 Process.join()  
 print('main')

进程池创建的子进程需要进行迭代调用，进程池对象有以下几种常用方法：

apply\_async(func, args=(), kwds={})：定义并启动进程，func为函数，args非关键字参数，kwds关键字参数。

apply()：与apply\_async()类似，但是会阻塞进程，使进程不能并发处理，只能一个个执行，一般不会使用。

close() ：关闭进程池（pool），进程池不再生成进程。

terminal()：结束工作进程，不在处理未处理的任务。

join()：主进程阻塞等待子进程的退出， join()方法要在close()方法之后使用。

## 控制外部进程

使用Python的subprocess模块可以对Python程序外部的进程进行控制，比如用Python打开以及关闭软件，控制windows系统的软件时，实际上就是执行DOS命令。

import subprocess  
 # 启动程序  
 subprocess.run(r'E:\常用软件\Notepad++\notepad++.exe')  
 # 关闭进程树  
 subprocess.run('taskkill /f /t /im notepad++.exe')

## 进程间通信

进程之间有时候需要进行通信，python提供了多种进程通信的方式，主要multiporcessing模块中的Queue和Pipe这两种方式，Queue用于多个进程间实现通信，Pipe是两个进程的通信。

**（1）Queue队列**

Queue主要有两个方法，put与get。Put方法：以插入数据到队列中。Get方法：从队列读取并且删除一个元素。

from multiprocessing import Process,Queue  
 from time import sleep  
 import random,os,datetime  
  
 def Write(Q):  
 for i in range(5):  
 sleep(random.random())  
 Q.put(i)  
 print('进程1:',os.getpid(),'往队列中写入数据:',i,datetime.datetime.now())  
 def Read(Q):  
 while True:  
 print('进程2:',os.getpid(),'从队列中取数据:',Q.get(),datetime.datetime.now())  
 if \_\_name\_\_ == '\_\_main\_\_':  
 Q=Queue()  
 Process1=Process(target=Write,args=(Q,),name='进程1')  
 Process2=Process(target=Read,args=(Q,),name='进程2')  
 Process1.start()  
 Process2.start()  
 Process1.join()  
 Process2.terminate()

**（2）Pipe管道**

Pipe常用于两个进程，两个进程分别位于管道的两端。Pipe方法返回（conn1,conn2）代表一个管道的两个端，Pipe方法有duplex参数，默认为True，即全双工模式，若为False，conn1只负责接收信息，conn2负责发送。发送信息的方法为send，接收信息的方法为recv。

from multiprocessing import Process,Pipe  
 from time import sleep  
 import datetime,os,random  
  
 def send(pipe):  
 for i in range(5):  
 sleep(random.random())  
 pipe.send(i)  
 print('进程1:',os.getpid(),'发送数据：',i,datetime.datetime.now())  
 def recv(pipe):  
 while True:  
 print('进程2:',os.getpid(),'接收数据：',pipe.recv(),datetime.datetime.now())  
 if \_\_name\_\_ == '\_\_main\_\_':  
 pipe=Pipe(duplex=False)  
 Process1=Process(target=send,args=(pipe[1],),name='进程1')  
 Process2=Process(target=recv,args=(pipe[0],),name='进程2')  
 Process1.start()  
 Process2.start()  
 Process1.join()  
 Process2.terminate()

# 线程

## 创建线程

多任务可以由多进程完成，也可以由一个进程内的多线程完成。线程是操作系统直接支持的执行单元，高级语言通常都内置多线程的支持，Python也不例外，Python提供的threading模块来多线程进行操作。

import threading,datetime,random  
 from time import sleep  
  
 def func1(Thread):  
 for i in range(5):  
 sleep(random.random())  
 print('线程',Thread,':',threading.current\_thread(),'正在执行',datetime.datetime.now())  
 def func2(Thread):  
 for i in range(5):  
 sleep(random.random())  
 print('线程', Thread, ':', threading.current\_thread(), '正在执行', datetime.datetime.now())  
 if \_\_name\_\_ == '\_\_main\_\_':  
 print(threading.current\_thread())  
 Thread1=threading.Thread(target=func1,args=('线程1',))  
 Thread2=threading.Thread(target=func2,args=('线程2',))  
 Thread1.start()  
 Thread2.start()  
 Thread1.join()  
 Thread2.join()

由于任何进程默认就会启动一个线程，该线程称为主线程，主线程又可以启动新的线程。threading模块有个current\_thread()函数，它返回当前线程的实例。主线程实例的名字叫MainThread，子线程的名字在可以在创建时指定，如果不起名字Python就自动给线程命名为Thread-1，Thread-2......

## 线程锁

多线程和多进程最大的不同在于，多进程中的同一个变量，各自有一份拷贝存在于每个进程中，互不影响。而多线程中，所有变量都由所有线程共享，任何一个变量都可以被任何一个线程修改。线程之间共享数据最大的危险在于多个线程同时改一个变量，造成数据混乱。

这种情况下可以给线程执行的函数上一把锁，上了锁之后的函数当某个线程在执行时，其他的线程不能执行该函数，只能等待锁被释放后其他线程才能执行。线程锁通过threading.Lock()来创建，通过锁的acquire()来获取锁，release()来释放锁。

import threading,datetime,random  
 from time import sleep  
  
 def func(Thread):  
 lock.acquire()  
 for i in range(5):  
 sleep(random.random())  
 print('线程',Thread,':',threading.current\_thread(),'正在执行',datetime.datetime.now())  
 lock.release()  
if \_\_name\_\_ == '\_\_main\_\_':  
 lock=threading.Lock()  
 print(threading.current\_thread())  
 Thread1=threading.Thread(target=func,args=('线程1',))  
 Thread2=threading.Thread(target=func,args=('线程2',))  
 Thread1.start()  
 Thread2.start()  
 Thread1.join()  
 Thread2.join()

## 线程作用域

ThreadLocal线程变量是一个全局变量，它可以把变量与线程捆绑在一起，各个线程之间的变量相互独立。ThreadLocal最大的用处就是线程执行多层嵌套函数时，参数不需要一层层传递，只要一开始把参数绑定在线程上，然后通过线程调用时来传递参数。

import threading  
  
 def fun1(name):  
 ThreadLocal.name=name  
 fun2()  
 def fun2():  
 print(threading.current\_thread().name,'的参数是',ThreadLocal.name)  
 if \_\_name\_\_ == '\_\_main\_\_':  
 ThreadLocal=threading.local()  
 Thread1=threading.Thread(target=fun1,args=('参数1',),name='线程1')  
 Thread2=threading.Thread(target=fun1,args=('参数2',),name='线程2')  
 Thread1.start()  
 Thread2.start()  
 Thread1.join()  
 Thread2.join()

结果为：线程1的参数是参数1，线程2的参数是参数2 。

# 正则表达式

https://www.cnblogs.com/shenjianping/p/11647473.html

正则表达式，又称规则表达式，正则表达式通常被用来检索、替换那些符合某个模式或者规则的文本。比如\d可以匹配一个数字，\w匹配字母或数字或下划线，.匹配除换行符以外的任意字符，\s匹配空白符。

要匹配变长的字符，在正则表达式中，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符。例如\d{3}\s+\d{3,8}：\d{3}表示匹配3个数字，例如'010'；\s+表示匹配至少一个空格；\d{3,8}表示3-8个数字，例如'1234567'。python使用正则表达式处理匹配字符串，需要导入re模块。re模块常用有以下方法：

**（1）re.compile（pattern，flags = 0）**

将正则表达式模式编译为正则表达式对象，编译后的对象可使用match()、search()等方法用于匹配字符串。pattern正则表达式字符串，falgs匹配模式，如下：

Regex=re.compile(r'\d+\w{2}')

**（2）re.search（pattern，string，flags = 0）**

扫描字符串以查找正则表达式模式产生匹配项的第一个位置，然后返回相应的match对象。pattern正则表达式，string匹配的目标字符串，flag匹配模式。如果字符串中没有与模式匹配，则返回None：

re.search(r'e\d{2}\w+','q12w3e45\_t67')

使用re.search（pattern，string，flags = 0）.group()可以返回匹配到的字符串结果，如果没有匹配到则调用group()函数时就会报错。

**（3）re.match（pattern，string，flags = 0）**

与search()函数区别在于匹配时要从字符串的第一个字符开始匹配，如果匹配不到则返回None：

re.match(r'e\d{2}\w','e45\_t67')

**（4）re.fullmatch（pattern，string，flags = 0）**

如果整个字符串与正则表达式模式匹配，则返回相应的match对象。

re.fullmatch(r'a\w{2}\d{3}','abc123').group()

**（5）re.split(pattern,string,maxsplit=0,flag=0)**

支持正则表达式的拆分函数，maxsplit为切割次数。

print(re.split('\W', 'Words, words, words.'))

\W为非字母数字下划线。

# 常用内建模块

## datetime时间

datetime是Python处理日期和时间的标准库。

**（1）date**

date表示日期，date.max/date.min返回最大/小日期，实例化方法为Date=date(year，month，day)：year的范围为[1,9999]，month的范围为[1,12]，day的范围为[1,31]。

date.today()：返回系统当前日期的实例化日期对象，日期对象可以访问日期属性以及使用日期方法。

Date.year/month/day：返回年/月/日。

Date.replace(2020,1,1)：返回一个新的日期对象，原日期对象不变。

Date.timetuple()：返回日期对应的time.struct\_time对象，对象信息包括年、月、日、时、分、秒、星期（0表示星期一）、一年中的第几天、是否夏令时。

Date.toordinal()：返回日期是是自 0001-01-01 开始的第多少天。

Date.weekday()/Date.isoweekday()：返回当前日期对应的星期数，第一个函数返回[0-6]，第二个函数返回[1-7]。

Date.isocalendar()：返回一个元组：（年份，这一天是这一年的第几个星期，星期几）。

Date.isoformat()：返回“YYYY-MM-DD”'格式的日期字符串。

Date.strftime：返回指定格式的日期。

**（2）time**

time表示时间，time.max/time.min返回最大/小时间，实例化时间的方法为Time=time(hour,minute,second,microsecond)：hour范围为[0,24]，minute范围为[0,59]，second范围为[0,59]，microsecond范围为[0, 1000000]。实例化时间对象可以访问时间属性以及使用时间方法：

Time.hour/minute/second：返回时间的时、分、秒。

Time.replace()：返回一个新的时间对象，原时间对象不变。

Time.isoformat()：返回一个HH:MM:SS.%f格式的时间字符串。

Time.strftime()：返回指定格式的时间字符串。

**（3）datetime**

datetime表示日期时间，实例化方法为Datetime=datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0)。

Datetime.today()：返回一个表示当前本期日期时间的datetime对象。

Datetime.now([TZ])：返回指定时区日期时间的datetime对象，如果不指定tz参数则结果同Datetime.today()。

Datetime可以访问日期、时间所拥有的属性。

Datetime.date()/time()：返回日期时间的日期、时间对象。

Datetime.strptime(string,format)：字符串时间转换为时间对象，string字符串格式的时间，format时间格式。

**（4）timedelta**

datetime模块提供的timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, hours=0, weeks=0)类可以对datetime.date、datetime.datetime对象做算术运算，这个差值的单位可以是：天、秒、微秒、毫秒、分钟、小时、周，例如：

Datetime+timedelta(weeks=1,days=1)

**（5）时间戳**

以1970年1月1日 00:00:00这个时间的偏移量叫做时间戳，表示离1970年1月1日 00:00:00这个时间过去了多少秒，时间戳通常用来做数据的唯一性标识。时间可以转换为时间戳，时间戳也可以转换为时间，转换的方法如下：

datetime.timestamp())：时间转换为时间戳。

datetime.fromtimestamp(1586003496))：时间戳转换为时间。

## collections容器数据

**（1）Counter计数器**

Counter计数器，用于统计元素的数量，常用的方法如下：

count=Counter(abddcaeabe'')：返回一个计数器对象，统计每个元素出现的次数，并且将结果按从大到小生成一个字典{'a': 3, 'b': 2, 'd': 2, 'e': 2, 'c': 1}，相同的统计次数先出现的元素在前，计数器对象可以使用字典对应的方法。

count.most\_common(n)：返回前n个重复次数最多的键值对，次数相同的取前一个。

count.update()：增加元素的重复次数，例如count.update('aaa')、count.update({'a':1,'b':2})。

count.subtract()：减少元素的重复次数，例如count.subtract('aa')、count.subtract({'a':1,'e':1})。

**（2）OrderDict有序字典**

数据结构字典Dict是无序的，OrderDict可以提供有序字典，顺序按照key插入的顺序。

Dict=OrderedDict()  
 Dict['name']='李明'  
 Dict['age']=20  
 Dict['address']='浙江省杭州市余杭区'

**（3）namedtuple命名元组**

namedtuple是一个函数，它用来创建一个自定义tuple对象，给tuple每个元素起一个名字，这样就可以通过名字来访问元组里的元素。

from collections import namedtuple  
 Tuple=namedtuple('info',['name','age','address'])  
 List=['张三',20,'浙江省杭州市余杭区']  
 Tuple(\*List)

结果为info(name=’张三’,age=20,address=’浙江省杭州市余杭区’)

**（4）deque双向队列**

deque双向队列其实就是double-ended queue的缩写，单向队列只能从一边放从另外一边取，双向队列两头都可以放也都可以取。实例化对象方法为Deque=deque(iterable)，deque双向很多方法和list的方法相同，以下为队列特殊的方法：

Deque.appendleft()：左增。

Deque.extendleft：左扩展，如果有多个元素，则依次往左扩展。

Deque.popleft()：左弹出。

**（5）defaultdict默认字典**

defaultdict默认字典，为字典值设置一个默认值或者类型，例如可以给字典值设置成列表类型，那么字典值将可以使用列表的方法：

DefaultDict=defaultdict(list)  
 people = [['male','winter'],['female','elly'], ['male','frank']]

for info in people:  
 DefaultDict[info[0]].append(info[1])  
 print(DefaultDict)

结果为：{'male': ['winter', 'frank'], 'female': ['elly']}

## itertools迭代器模块

itertools 是python的迭代器模块。

**（1）count(start,step=none)无限数列迭代器**

count()可以生成一个以start开始每次增长step（step不设置时默认为1）的无限数列迭代器，对迭代器进行迭代时如果不强制停止则会一直迭代下去。

Iterable=count(1,3)  
 for i in Iterable:  
 if i>20:  
 break  
 print(i)

结果为：1，4，7，10，13，16，19。

**（2）cycle(iterable)无限循环迭代器**

cycle()可以把iterable变成无限循环迭代的迭代器，这样可以对迭代器进行多伦迭代。

Iterable=('ab','cd','ef')  
 Cycle=cycle(Iterable)  
 count=len(Iterable)\*2  
 i=0  
 for iter in Cycle:  
 i+=1  
 print(iter)  
 if i==count:  
 break

**（3）repeat(object,times=None)重复迭代器**

repeat()生成一个重复times次object的迭代器，若times=None则无限次重复。

Repeat=repeat((1,2,3),2)  
 for i in Repeat:  
 print(i)

结果为：（1,2,3），（1,2,3）。

**（4）chain(\*iterable)拼接迭代器**

chain()将iterable的元素进行拼接后生成迭代器，按iterable元素迭代顺序进行拼接。

Chain=chain([1,2,3,4],('a','b','c','d'))  
 for i in Chain:  
 print(i)

结果为：1,2,3,4,a,b,c,d。

**（5）islice(iterable,start,end,step)切片迭代器**

islice()对terable进行切片，start是开始索引，end是结束索引，step是间隔，start开end闭。

List=list(islice(range(20),1,19,2))  
 print(List)

结果为[1,3,5,7,9,11,13,15,17]。

## urllib网页请求模块

urllib是一个高级的web交流库，其核心功能就是模仿web浏览器等客户端，去请求相应的资源，并返回一个类文件对象。urllib支持各种web协议（HTTP、FTP、Gopher），同时也支持对本地文件进行访问，urllib库有四大模块：request、error、parse、robotparser。

**（1）request请求模块**

http请求模块，可以用来模拟发送请求。就好比在浏览器中输入网址然后回车一样，只需要给库方法传入URL以及额外的参数，就可以模拟实现这个过程。

request.Request(url,data=None,headers={},method=None)：根据数据生成一个请求报文，生成的报文可以被request.urlopen()方法使用。

url：请求的URL，必须传递的参数，其他都是可选参数。

data：上传的数据，必须传bytes字节流类型的数据，自定义的数据需要使用bytes()方法进行转换。如果它是字典，需要先使用urllib.parse模块里的urlencode()进行编码。

headers：它是一个字典，传递的是请求头数据，也可以通过调用请求实例的方法add\_header()来添加。

method：是一个字符串，用来指示请求使用的方法，如：GET,POST,PUT。

request.urlopen(url,data=None,[timeout,],cafile=None,capath=None,cadefault=False,context=None)：函数用于实现对目标url的访问，并返回HTTPResponse类型的对象。

url：请求的URL。

data：上传的数据，必须传bytes字节流类型的数据，如果它是字典，可以用urllib.parse模块里的urlencode()编码。urlopen默认请求方法是get，如果传递data参数，urlopen将使用post方法请求。

timeout参数：用于设置超时时间，单位为秒，如果请求超出了设置时间还未得到响应则抛出异常。

Cafile、capath、cadefault：用于实现可信任的CA证书的HTTP请求。

context参数：实现SSL加密传输。

from urllib import request,parse  
info=request.Request(method='GET',url='http://192.168.100.233:7005')  
response1=request.urlopen(info)  
Dict={'username':'admin','password':'000000'}  
data=bytes(parse.urlencode(Dict).encode('utf-8'))  
response2=request.urlopen('http://192.168.100.233:7005/index\_zt.jsp',data=data)

使用request.urlopen()方法进行http请求，将会返回一个HTTPResponse类型的对象，通过调用对象的方法及属性可以访问响应信息。

response.read().decode('utf-8') #返回网页内容

response.getheader('server') #返回响应头中的server值

response.getheaders() #以列表元祖对的形式返回响应头信息

response.fileno() #返回文件描述符

response.version #返回版本信息

response.status) #返回状态码200，404代表网页未找到

response.debuglevel #返回调试等级

response.closed #返回对象是否关闭布尔值

response.geturl() #返回检索的URL

response.info() #返回网页的响应头信息

response.getcode() #返回响应的HTTP状态码

response.msg #访问成功则返回ok

response.reason #返回状态信息

**（2）error错误处理模块**

error模块定义了由request模块产生的异常，如果出现问题，request模块便会抛出error模块中定义的异常。

error.URLError：它继承自OSError类，是error异常模块的基类，由request模块产生的异常都可以通过捕获这个类来处理。它只有一个属性reason，即返回错误的原因。

from urllib import error,request  
 try:  
 response=request.urlopen('http://localhost:9087')  
 except error.URLError as E:  
 print(E)  
 print(E.reason)

error.HTTPError：它是URLError的子类，专门用来处理HTTP请求错误，比如认证请求失败，它有3个属性：

code：返回HTTP的状态码，如404页面不存在，500服务器错误等。

reason：同父类，返回错误的原因。

headers：返回请求头。

**（3）parse解析模块**

parse模块定义了处理URL的标准接口，例如实现URL各部分的抽取，合并以及链接转换。

parse.urlparse(urlstring,scheme='',allow\_fragments=True)：解析一个url。

urlstring：待解析的URL，字符串。

scheme：它是默认的协议，如http或者https，URL如果不带http协议，可以通过scheme来指定，如果URL中制定了http协议则URL中生效。

allow\_fragments：是否忽略fragment即锚点，如果设置为False，fragment部分会被忽略，反之不忽略。

parse.urlparse()方法返回一个ParseResult对象，内容实现URL的识别和分段，分别是scheme(协议)、netloc(域名)、path(路径)、params(参数)、query(查询条件)、fragment(锚点)。可以使用ParseResult对象的属性进行访问这些信息，也可以通过索引进行访问。

from urllib import parse  
result=parse.urlparse('http://www.baidu.com/index.html;user?id=5#comment')  
print(type(result),result,sep='\n')  
print(result.scheme,result[0])  
print(result.netloc,result[1])  
print(result.path,result[2])  
print(result.params,result[3])  
print(result.query,result[4])  
print(result.fragment,result[5])

parse.urlunparse()：与urlparse()相反，通过列表或者元组的形式接受一个可迭代的对象，实现URL构造。列表或者元组的元素必须有6个，对应url=的识别信息。

List=['https','128.8.18.233:7001','/investment/invest/contract.jsp','','book\_code=1&contract\_id=200','']  
url=parse.urlunparse(List)  
print(url)

结果https://128.8.18.233:7001/investment/invest/contract.jsp?book\_code=1&contract\_id=200

parse.urlsplit()：与urlparse()方法类似，它会返回5个部分，把params合并到path中。

parse.urlunsplit()：与urlunparse()类似，唯一的区别是长度必须是5个，它省略了params。

urljoin(base\_url,url)：通过将基本URL（base）与另一个URL(url)组合起来构建完整URL，它会使用基本URL组件，协议(schemm)、域名(netloc)、路径(path)、来提供给URL中缺失的部分进行补充，最后返回结果。base\_url提供了三项内容scheme,netloc,path，如果这3项在url中不存在就给予补充，如果url中存在则不补充，而base\_url中的params,query和fragment是不起作用的。

URL=parse.urljoin('https://127.0.0.1:9081/project/forms/invest.jsp?busi\_id=120305&invest\_id=90','http://128.8.18.233:9080/investment/invest/contract.jsp?product\_id=1000&contract\_id=20')  
url=parse.urljoin('http://128.8.18.233:9080/investment/invest/contract.jsp','?product\_id=1000&contract\_id=20')  
print(url)  
print(URL)

结果为：

http://128.8.18.233:9080/investment/invest/contract.jsp?product\_id=1000&contract\_id=20

urlencode(dict)：在构造GET请求参数时很有用，它可以将字典转化为GET请求参数。

Dict={'product\_id':1000,'contract\_id':20}  
url='http://128.8.18.233:9080/investment/invest/contract.jsp?'+ parse.urlencode(Dict)  
print(url)

结果为：

http://128.8.18.233:9080/investment/invest/contract.jsp?product\_id=1000&contract\_id=20

parse\_qs()：与urlencode()正好相反，它是用来反序列化的，如将GET参数转换回字典格式。

Dict=parse.parse\_qs('product\_id=1000&contract\_id=20')  
print(Dict)

结果为：{'product\_id': ['1000'], 'contract\_id': ['20']}。

parse\_qsl()：将参数转换为元组组成的列表。

List=parse.parse\_qsl('product\_id=1000&contract\_id=20')  
print(List)

结果为：[('product\_id', '1000'), ('contract\_id', '20')]。

quote()：按照标准，URL只允许一部分ASCII字符（数字字母和部分符号），其他的字符或者汉字是不符合URL标准的，所以URL中使用其他字符就需要进行URL编码，quote()函数提供了此功能。

value=parse.quote('汉字')  
url='http://127.0.0.1:9080//investment/invest/contract.jsp?name='+value  
print(url)

结果为：http://127.0.0.1:9080//investment/invest/contract.jsp?name=%E6%B1%89%E5%AD%97

unquote()：与quote()相反，用来进行URL解码 。

print(parse.unquote(value))

# 常用第三方模块

## requests

requests比urllib模块更简洁，支持HTTP连接保持和连接池，支持使用cookie保持会话，支持文件上传，支持自动响应内容的编码，支持国际化的URL和POST数据自动编码。 使用Requests可以轻而易举的完成浏览器可有的任何操作，requests会自动实现持久连接keep-alive，requests常用get与post请求。

1. requests.get(url，params，kwargs)get请求。

url：访问地址。

param：参数。

kwargs：其他数据，如果全部放在一起则需要整理成字典格式。

requests.get('http://192.168.100.233:7003')

get请求可以添加参数，参数可以放在URL里面，URL问号后面的内容就是参数以及参数值，多个参数之间用&隔开。参数也可以放在params，格式为字典或者json，这两种效果实现是一样的。

requests.get('http://192.168.100.233:7003/transaction/project/product\_set.jsp?product\_id=856&contract\_id=200')  
requests.get('http://192.168.100.233:7003/transaction/project/product\_set.jsp?',params={'product\_id':856,'contract\_id':200})

这两个访问的地址都是：[http://192.168.100.233:7003/transaction/project/product\_set.jsp?pro](http://192.168.100.233:7003/transaction/project/product_set.jsp?prod)

[d](http://192.168.100.233:7003/transaction/project/product_set.jsp?prod)uct\_id=856&contract\_id=200

（2）requests.post(url，data，json，kwargs)post请求。

url：请求地址。

data：字典格式的请求数据。

json：json格式的请求数据。

kwargs：其他数据，如果全部放在一起则需要整理成字典格式。

response=requests.post('http://192.168.100.233:7003/',data={'username':'admin','password':'000000'})

（3）响应内容的获取

response.encoding：响应内容的编码格式，也可以对响应内容设置编码格式（response.encoding=’utf-8’）。

response.text：字符串方式的响应体，会自动根据响应头部的字符编码进行解码。

response.content：以字节形式（二进制）返回，字节方式的响应体，会自动为你解码 gzip 和 deflate 压缩。

response.headers：以字典对象存储服务器响应头，若不存在则返回None。

response.status\_code：响应状态码（200）。

response.raw：返回原始响应体，也就是urllib的response对象。

response.ok：返回布尔值，表示请求是否成功。

response.json：Requests中内置的JSON解码器，以json形式返回,前提返回的内容确保是json格式的，不然解析出错会抛异常。

response.raise\_for\_status()：失败请求(非200响应)抛出异常。

（4）定制请求头与cookie信息。

通过参数headers可以设置请求头信息，请求头信息是一个字典。

requests.post('http://localhost:7005/',headers={'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'},data={'username':'admin','password':'000000'})

通过参数cookie可以设置请求的cookie信息，通过cookie信息可以绕过登录的操作。

response1=requests.post('http://localhost:7005/',headers={'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'},data={'username':'admin','password':'000000'})  
List=response1.headers['Set-Cookie'].split('=',1)  
Data={'book\_code':1,'product\_id':855,'proj\_id':'TZ\_451','checker\_empno':6393,'checker\_empno\_name':6393,'proj\_check\_seq':'1111','proj\_check\_comnt':'1111','checker\_mem\_role':'1111','checker\_mem\_check\_comnt':'1111','reg\_person':'1111','reg\_org':'1111','reg\_tm\_picker':'2020-04-11','reg\_tm':20200411,'checker\_mem\_name':'陈立群'}  
Cookie={List[0]:List[1]}  
response2=requests.post('http://localhost:7005/jtinfo/tproj\_check\_info\_update.jsp?book\_code=1&product\_id=855',headers={'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'},cookies=Cookie,data=Data)

## chardet

在处理字符串时，常常会遇到不知道字符串是何种编码，如果不知道字符串的编码就不能将字符串转换成需要的编码，chardet是一个非常优秀的编码识别模块，可以帮助识别字符编码。调用chardet.detect(byte\_str)，将会返回encoding字符编码、confidence检测正确性、language语言三种信息组成的一个字典，例如：

chardet.detect(b'Hello, world!')

结果为：{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}

byte\_str参数必须是字节类型（bytes）字符串，python有两种不同的字符串，一种是文本字符串，另一种是字节字符串。对于文本字符串python内部采用unicode存储，而字节字符串显示原始字节序列或者ASCII。实例化一个字符串会得到一个str对象，如果想得到bytes，那就在文本之前加上前缀b（必须是ASCLL码才行，其他只能用encode），或者encode一下。所以，str对象有一个encode方法，bytes对象有一个decode方法。str与bytes之间可以进转换，str转换成bytes只需要在str前加b或者调用str对象的encode(编码格式)方法设置其编码格式，byes转换为str只需要调用bytes的decode()就可以转换为str。

Bytes='中文'.encode('UTF-8')  
 Str=Bytes.decode()

# 邮件发送

SMTP是发送邮件的协议，Python内置对SMTP的支持，可以发送纯文本邮件、HTML邮件以及带附件的邮件。Python对SMTP支持有smtplib和email两个模块，email负责构造邮件，smtplib负责发送邮件。

## email

通过email模块中的MIMEMultipart类的实例化对象来组成邮件的内容，一般邮件内容由两个部分组成，第一部分是头部信息，头部信息包含邮件标题、发件人标识、收件人、抄送人等，这些信息需要通过Header方法来封装。

from email.mime.multipart import MIMEMultipart  
from email.header import Header  
# 实例化邮件内容对象。  
Email=MIMEMultipart()  
# 添加头部信息的标题信息。  
Email['Subject']=Header('邮件标题')  
# 添加发件人标识。  
Email['From']='Python自动发'  
# 添加收件人，多个收件人需要用逗号进行隔开。  
Receivers=['583870243@qq.com','1359881082@qq.com']  
Email['to']=Header(','.join(Receivers))  
# 添加抄送人，多个抄送人需要用逗号进行隔开。  
Chaosongs=['583870243@qq.com','1359881082@qq.com']  
Email['Cc']=Header(','.join(Chaosongs))

第二部分是邮件的正文信息，通过email的MIMEText可以生成邮件的正文信息，生成的邮件正文需要添加到实例化邮件对象中。

from email.mime.text import MIMEText

# 添加正文内容。  
msg='我是正文内容'  
Zhengwen=MIMEText(msg,'plain','utf-8')  
Email.attach(Zhengwen)

## smtplib

组装好的邮件内容，通过smtplib模块进行发送，发送邮件需要实例化SMTP对象，然后再调用对象的方法进行发送，实例化方法为smtplib.SMTP(host,port,local\_hostname)：

Host：SMTP服务器主机地址，如QQ邮箱的SMTP服务器主机地址为smtp.qq.com；port：指定SMTP服务使用的端口号，一般情况下SMTP端口号为25。local\_hostname: 如果SMTP在你的本机上，你只需要指定服务器地址为localhost即可。

from smtplib import SMTP

# 实例化SMTP协议对象。  
Smtp\_server='smtp.qq.com'  
Smtp\_port=25  
Smtp=SMTP(Smtp\_server,Smtp\_port)

实例化好SMTP对象后，调用对象的login(user,password)方法进行登录，登录的用户名为邮箱地址，密码为SMTP服务的服务码，SMTP服务的服务码在向相应的邮箱提供商申请后可获得。

User='2821644642@qq.com'  
Password='zlwmyfxysgjpdech'  
Smtp.login(User,Password)

登录之后就可以调用SMTP对象的sendmail(from\_addr,to\_addrs,main\_options)方法进行发送邮件：from\_addr为发送人邮箱，to\_addrs为收件人（收件人为收件人与抄送人），main\_options为邮件内容。

# 发送邮件。  
Smtp.sendmail(User, Receivers+Chaosongs, Email.as\_string())

发送完邮件后，进行退出。

# 退出邮箱。  
Smtp.quit()

# 连接数据库

## MSSQL

连接MSSQL需要下载安装第三方包pymssql，pymssql包有点特殊，使用pip包下载安装时会报错，只能网上下载对应版本的包然后进行安装（地址为：https://www.lfd.uci.edu/~gohlke/pythonlibs/#pymssql）。

pymssql连接数据库基本操作流程：（1）使用connect创建连接对象；（2）使用connect.cursor()创建游标对象，SQL语句的执行基本都在游标上进行；（3）使用游标的方法cursor.executeXXX执行SQL语句，cursor.fetchXXX获取查询结果；（4）调用close方法关闭游标cursor和数据库连接。

pymssql.connect(server, user, password, database)创建数据库连接对象：server为数据库的服务器名称或者ip，user为数据库账号，password为数据库账号的密码，database为连接的数据库。

import pymssql  
# 创建数据库链接  
sqlserver=pymssql.connect('127.0.0.1:1433','sa','000000','intrust')

使用数据库连接对象的cursor()创建游标来执行SQL语句。

# 创建游标  
Cursor=sqlserver.cursor()

使用cursor()对象的execute()方法执行sql语句。

# 执行数据库数据库语句  
Cursor.execute('''  
 if not exists(select 1 from sysobjects where name='Student')begin  
 create table Student(  
 Serial\_no integer not null identity,  
 Name nvarchar(6),  
 Age integer,  
 Address nvarchar(30)  
 )  
 end  
''')  
Cursor.execute('''  
 if not exists(select 1 from Student where Name in('小明','小红','小刚'))begin  
 insert into Student(Name,Age,Address)  
 values('小明',20,'北京市朝阳区'),  
 ('小红',21,'上海市徐汇区'),  
 ('小刚',22,'杭州市余杭区')  
 end  
''')

对数据库的增删改都必须提交事务，否则不生效，使用数据库连接对象的commit()方法来提交事务，如果数据库执行时发生错误，则可以使用数据库连接对象的rollback()来回滚事务。

# 提交事务  
sqlserver.commit()

# 回滚事务

sqlserver.rollback()

如果是查询数据信息，则查询结果会存入到游标中，读取数据需要使用游标对象的fetchone()（每次取一个）方法或者是fetchall()（全部取出来）把数据取出来。

# 查询信息  
Cursor.execute('select \* from Student')  
student=Cursor.fetchone()  
while student:  
 print(student)  
 student = Cursor.fetchone()

## MYSQL

MYSQL用法与MSSQL类似，差别在于MYSQL与MSSQL自身上的差异。使用commit()提交事务的时候需要注意，如果执行多条增删改语句时有部分语句执行报错，那么commit()会提交成功的事务，这种情况下需要用rollback()方法进行回滚，所以执行数据库的增删改语句时最好是用try执行，然后使用except捕获异常后处理rollback()。

import pymssql  
try:  
 SQL=pymssql.connect('127.0.0.1:1433','sa','000000','TEST')  
 CURSOR=SQL.cursor()  
 CURSOR.execute("update Student set Name='小蓝' where Serial\_no=1")  
 CURSOR.execute("update Student set Address='广州市天河区' where Serial\_no=2")  
 CURSOR.execute("update Student set Age=50 where Serial\_no=3")  
except Exception as E:  
 print(E)  
 SQL.rollback()  
else:  
 SQL.commit()  
CURSOR.execute('select \* from Student')  
STUDENT=CURSOR.fetchall()  
print(STUDENT)  
CURSOR.close()  
SQL.close()

数据库连接创建好最好需要关闭，游标也需要关闭，所以可以使用with的写法来编写。

import pymssql  
with pymssql.connect('127.0.0.1:1433','sa','000000','TEST') as SQL:  
 with SQL.cursor() as CURSOR:  
 CURSOR.execute('select \* from Student')  
 Student=CURSOR.fetchall()  
 print(Student)