# 1 入门部分

## 1.1 基本类型和操作

### 1.1.1 基本类型(010简单类型)

# 1.1 hello world程序，字符串类型  
# Python会忽略注释，你可以用它们来写程序注解，或提醒自己代码试图完成的事。  
# 这一行中，#标志之后的所有文本都是注释。  
print('Hello world!')   
print("Hello world!")  
# 1101 这是一个字符串类型str  
# 所有用单引号（'）、双引号（"）或成组的3个引号（单引号或双引号）包围且没有前缀的值都表示str数据类型：  
# 1102 print是python的内置输出函数。  
# 1103代码行print('Hello world!')表示“打印出字符串'Hello world!'的文本”。  
# Python执行到这行时，你告诉Python调用print()函数，并将字符串“传递”给函数。  
# 传递给函数的值称为“参数”。  
# 请注意，引号没有打印在屏幕上。它们只是表示字符串的起止，不是字符串的一部分。  
  
# 1.2 数值计算，整数型和浮点类型  
print(2+2) # 整数相加  
print(-2+2) # 整数相加  
print(4-2) # 整数相减  
print(4\*2) # 整数相乘  
print(4/2) # 整数相除，结果出现了不一样，因为整数相除的结果是一个浮点类型的结果  
# 1201 整型（或int）数据类型表明值是正负整数  
# 1202 带有小数点的数据类型，如3.14，称为“浮点型”（或float）  
print(12.51 + 11.37)  
print(12.5 + 11.37)  
# 1203 浮点数的计算精度问题  
# 这是因为小数以二进制形式表示时的有穷性导致的。  
# 我们知道，将一个小数转化为二进制表示的方式是，不断的乘2，取其中的整数部分。例如：  
# (1) 0.625\*2 = 1.25, 整数部分为1，小数部分为0.25   
# (2) 0.25 \* 2 = 0.5 , 整数部分为0，小数部分为0.5   
# (3) 0.5 \* 2 = 1 , 整数部分为1，小数部分为0   
# 所以0.625的二进制表示就是0.101。  
# 然而有些小数，例如0.4，并不能够精确的转化为二进制表示，用上面的这种方法计算：  
# (1) 0.4\*2=0.8 整数部分为0，小数部分为0.8   
# (2) 0.8\*2=1.6 整数部分为1，小数部分为0.6   
# (3) 0.6\*2=1.2 整数部分为1，小数部分为0.2   
# (4) 0.2\*2=0.4 整数部分为0，小数部分为0.4   
# (5) 0.4\*2=0.8 整数部分为0，小数部分为0.8  
# (6) 0.8\*2=1.6 整数部分为1，小数部分为0.6   
# (7) 0.6\*2=1.2 整数部分为1，小数部分为0.2   
# 所以0.4转化为二进制，应该是0.0110... 这样一个无限循环小数。  
# 计算机的内存、cpu寄存器等等这些硬件单元都是有限的，只能表示有限位数的二进制位，因此存储的二进制小数就会和实际转换而成的二进制数有一定的误差。  
# 这个不是python 的问题，所有基于二进制的浮点数都会有这个问题  
# 原因在于大部分浮点数转换为二进制后都是无限循环小数，而浮点数不可能用无限大的内存来储存，所以会有舍入的误差。  
# 所以在python中不建议直接将两个浮点数进行大小比较，或者做精确的计算，往往会得到意想不到的结果。  
  
# 1.3 字符串计算 仅支持加法和乘法运算  
print('Hello' + 'world') # 输出会很难看，因为字会连在一起  
print('Hello' + ' ' + 'world') # 输出会很难看，因为字会连在一起  
print('Hello ' \* 3) # 使用字符串乘法将字符串复制三次  
  
# 1.4 字符串类型与整数和浮点型不能做运算  
print('Hello world!' + 12)  
# 1401 引发 发生异常: TypeError，异常是程序书写不符合运算或者语法导致的

### 1.1.2 基本类型(015字符串处理)

# 如需要输出这样的句子"我是学生"的英语为：I'm student.  
# "我是学生"的英语为：I'm student.  
  
# 单引号，双引号无论哪一种都无法解决  
# print(""我是学生"的英语为：I'm student.")  
# print('"我是学生"的英语为：I'm student.')  
  
# 转义字符  
# “转义字符”让你输入一些字符，它们用其他方式是不可能放在字符串里的。  
# 转义字符包含一个倒斜杠（\），紧跟着是想要添加到字符串中的字符。  
# （尽管它包含两个字符，但大家公认它是一个转义字符。）  
  
# \' 单引号   
# \" 双引号   
# \t 制表符   
# \n 换行符   
# \\  倒斜杠  
  
print("\"我是学生\"的英语为：I'm student.")  
print('"我是学生"的英语为：I\'m student.')  
  
print('\t"我是学生"的英语为：I\'m student.')  
print('"我是学生"的英语为：\nI\'m student.')  
# 在字符串开始的引号之前加上r，使它成为原始字符串。  
print(r"\"我是学生\"的英语为：I'm student.")  
print(r'"我是学生"的英语为：I\'m student.')  
  
# 字符串的in和not in操作符  
# 用in或not in连接两个字符串得到的表达式，将求值为布尔值True或False。  
# 这些表达式测试第一个字符串（精确匹配，区分大小写）是否在第二个字符串中。  
print("我是" in "我是学生")  
print("我是" not in "我是学生")  
  
# 字符串方法startswith()和endswith()  
# 如果它们所调用的字符串以该方法传入的字符串开始或结束，startswith()和endswith()方法返回True。  
# 否则，方法返回False。  
print("我是学生" .startswith("我是"))  
print("我是学生" .endswith("我是"))  
  
# 字符串的大小写方法upper()、lower()  
print('Hello world!'.upper())  
print('Hello world!'.lower())  
  
# strip()、rstrip()和lstrip()删除空白字符  
# strip()字符串方法将返回一个新的字符串，它的开头或末尾都没有空白字符。  
# lstrip()和rstrip()方法将相应删除左边或右边的空白字符。  
print(' Hello world! ')  
print(' Hello world! '.strip())  
print(' Hello world! '.rstrip())  
print(' Hello world! '.lstrip())

### 1.1.3 基本类型(020变量定义)

# 1.1 在变量(varible)中保存值  
var = 'Hello world!'  
print(var)  
  
VAR = 'Hello china!'  
print(VAR)  
# 1101 “变量”就像计算机内存中的一个盒子，其中可以存放一个值。  
# 用“赋值语句”将值保存在变量中。赋值语句包含一个变量名、一个等号（称为赋值操作符），以及要存储的值。  
  
# 变量名只能包含字母、数字和下划线。  
# 变量名不能以数字开头。  
# 变量名是区分大小写的  
  
# 1.2 变量的类型由存储的值来决定  
var = 'Hello world!'  
print(type(var)) # 使用type内置函数查看变量的类型，str是string的缩写，代表字符串类型  
var = 4  
print(type(var)) # 使用type内置函数查看变量的类型，int是integer的缩写，代表整数类型  
var = 4.1  
print(type(var)) # 使用type内置函数查看变量的类型，float代表浮点类型  
# 1201 第一次存入一个值，变量就被“初始化”（或创建)  
# 如果变量被赋了一个新值，老值就会在新值中被替代  
  
# 1.3 变量存储计算的结果  
varOne = 100  
varTwo = 5  
varResult = varOne / varTwo  
print(varResult)  
print(type(varResult))  
  
varOne = "Hello"  
varTwo = "World!"  
varResult = varOne + " " + varTwo  
print(varResult)  
print(type(varResult))  
# 1301 如果你的程序稍后将用到一个已求值的表达式的结果，就可以将它保存在一个变量中。

### 1.1.4 简单交互(040输入输出)

# 1.1 input()函数接受用户的输入  
age = input()  
print(age)  
print(type(age))  
# 1101input()函数调用是一个表达式，它求值为用户输入的任何字符串。  
# 函数等待用户在键盘上输入一些文本，并按下回车键。  
  
# 1.2 构造友好的输入输出  
age = input("请输入您的年龄：")  
print("你输入的年龄数字是：" + age)  
print("程序age的数据类型是：" + str(type(age)))  
  
# 1.3 交互式增强年龄程序  
age = input("请输入您的年龄：")  
print("你的年龄是：" + age)  
afterYears = input("请输入过去的年份：")  
ageResult = int(age) + int(afterYears)  
sentence = "过去" + afterYears + "年后，" + "你的年龄是" + str(ageResult) + "岁。"  
print(sentence)

### 1.1.5 类型转换(045类型转换)

# 1.1 整数类型转字符串类型  
age = 45  
wordsBegin = "I am "  
wordsEnd = " years old."  
# 想输出"I am 45 years old."，使用 wordsBegin + 45 + wordsEnd会报错  
sentence = wordsBegin + str(age) + wordsEnd  
print(sentence)  
# str()、int()和float()函数将分别求值为传入值的字符串、整数和浮点数形式。  
  
# 1.2 字符串类型转整数类型  
age = "45"  
afterYears = 10  
ageResult = int(age) + 10  
sentence = 'After 10 years, ' + wordsBegin + str(ageResult) + wordsEnd  
print(sentence)  
print('After 10 years, ' + wordsBegin + str(int(age) + 10) + wordsEnd)  
# str()、int()和float()函数将分别求值为传入值的字符串、整数和浮点数形式。

### 1.1.6 逻辑运算(050逻辑操作)

#1.1 “布尔”数据类型  
boolType = True  
print(boolType)  
boolType = False  
print(boolType)  
# 1101 虽然整型、浮点型和字符串数据类型有无数种可能的值，但“布尔”数据类型只有两种值：True和False。  
# Boolean（布尔）的首字母大写，因为这个数据类型是根据数学家George Boole命名的。  
# 在作为Python代码输入时，布尔值True和False不像字符串，两边没有引号，它们总是以大写字母T或F开头，后面的字母小写。  
  
# 1.2 比较操作符  
print(2 == 2)  
print(2 == 3)  
print('yes' == 'yes')  
print('Yes' == 'yes')  
  
boolType = 2 != 2  
print(boolType)  
print(2 != 3)  
print('yes' != 'yes')  
print('Yes' != 'yes')  
# 如果两边的值一样，==（等于）求值为True。如果两边的值不同求值为False。  
# ==和!=操作符实际上可以用于所有数据类型的值。  
# 整型或浮点型的值永远不会与字符串相等。表达式42 == '42'求值为False是因为，Python认为整数42与字符串'42'不同。  
  
print(2 >= 2)  
print(2 <= 3)  
  
print(2 > 2)  
print(2 < 3)  
# <、>、<=和>=操作符仅用于整型和浮点型值  
  
# 1.3 逻辑运算符  
result = True and True  
print(result)  
result = True and False  
print(result)  
result = False and False  
print(result)  
# 1301 如果两个布尔值都为True，and操作符就将表达式求值为True，否则求值为False。  
  
result = True or True  
print(result)  
result = True or False  
print(result)  
result = False or False  
print(result)  
# 1302 只要有一个布尔值为真，or操作符就将表达式求值为True。如果都是False，所求值为False。  
  
result = not True  
print(result)  
result = not False  
print(result)  
# 1303 和and和or不同，not操作符只作用于一个布尔值（或表达式）。not操作符求值为相反的布尔值。

## 1.2 程序控制语句

### 1.2.1 控制语句-010判定if

# 1.1if语句-判定  
# if语句的子句（也就是紧跟if语句的语句块），将在语句的条件为True时执行。如果条件为False，子句将跳过。  
age = 40  
if age == 40:  
 print("年龄为40")  
# if关键字；  
# 条件（即求值为True或False的表达式）；  
# 冒号；  
# 在下一行开始，缩进的代码块（称为if子句）。  
  
# 1.2 else语句  
# if子句后面有时候也可以跟着else语句。只有if语句的条件为False时，else子句才会执行。  
# 在英语中，else语句读起来可能是：“如果条件为真，执行这段代码。否则，执行那段代码”。  
age = 50  
if age == 40:  
 print("年龄为40")  
else:  
 print("年龄不为40")  
  
# 1.3 elif语句  
# 虽然只有if或else子句会被执行，但有时候可能你希望，“许多”可能的子句中有一个被执行。  
# elif语句是“否则如果”，总是跟在if或另一条elif语句后面。  
# 它提供了另一个条件，仅在前面的条件为False时才检查该条件。  
age = 50  
if age == 40:  
 print("年龄为40")  
elif age == 50:  
 print("年龄为50")  
else:  
 print("年龄不为40")  
  
# 1.4 if使用比较操作符和逻辑运算符混合运算  
age = 50  
if (age == 40) or (age == 50):  
 print("年龄为40或者50")  
else:  
 print("年龄不为40或者50")  
  
# 1.2 改进的年龄计算程序  
age = input("请输入您的年龄：")  
print("你的年龄是：" + age)  
if (int(age) <1) or (int(age) > 100) :  
 print("年龄输入错误！不能小于1岁大于100岁")  
else:  
 afterYears = input("请输入过去的年份：")  
 ageResult = int(age) + int(afterYears)  
 if ageResult > 100:  
 print("过去的年份输入错误！不能大于100岁")  
 else:  
 sentence = "过去" + afterYears + "年后，" + "你的年龄是" + str(ageResult) + "岁。"  
 print(sentence)  
  
# 1.3 增加变量对年龄程序的改进  
age = input("请输入您的年龄：")  
ageMax = 100  
print("你的年龄是：" + age)  
intAge = int(age) # 使用intAge变量将三次类型转换减少到一次  
if (intAge < 1) or (intAge > ageMax):  
 print("年龄输入错误！不能小于1岁或大于100岁")  
else:  
 afterYears = input("请输入过去的年份：")  
 ageResult = intAge + int(afterYears)  
 if ageResult > ageMax:  
 print("过去的年份输入错误！不能大于100岁")  
 else:  
 sentence = "过去" + afterYears + "年后，" + "你的年龄是" + str(ageResult) + "岁。"  
 print(sentence)

### 1.2.2 控制语句-020循环while

# 1.1while语句-循环  
# 利用while语句，可以让一个代码块一遍又一遍的执行。  
# 只要while语句的条件为True，while子句中的代码就会执行。在代码中，while语句总是包含下面几部分：  
  
# 关键字；  
# 条件（求值为True或False的表达式）；  
# 冒号；  
# 从新行开始，缩进的代码块（称为while子句）  
# while True:  
# print("开始循环了")  
# 使用shift+F5强制结束程序  
# 无限循环的程序的应用场景，监控系统中的连接探测  
  
# import time  
# while True:  
# print("开始循环了")  
# time.sleep(5)  
# time sleep() 函数推迟调用线程的运行，可通过参数secs指秒数，表示进程挂起的时间。  
  
# 1.2 使用变量来控制循环次数  
count = 0  
while count < 6:  
 count = count + 1  
 print("count的值为：" + str(count))  
# 使用变量来控制循环次数  
  
# 1.3 使用用户输入和break来控制循环  
# 有一个捷径，让执行提前跳出while循环子句。  
# 如果执行遇到break语句，就会马上退出while循环子句。  
while True:  
 name = input("请输入姓名(退出请输入exit)：")  
 if name == "exit":  
 break  
 else:  
 print("您输入的姓名是：" + name)  
  
# 1.3 使用用户输入和continue来控制循环  
# continue语句用于循环内部。  
# 如果程序执行遇到continue语句，就会马上跳回到循环开始处，重新对循环条件求值（这也是执行到达循环末尾时发生的事情）。  
while True:  
 name = input("请输入姓名(退出请输入exit)：")  
 if name == "exit":  
 break  
 else:  
 print("您输入的姓名是：" + name)  
 while True:  
 password = input(name + "请输入用户密码(退出请输入exit)：")  
 if password == "123456":  
 print("用户密码输入正确！")  
 break  
 else:  
 continue  
  
# 1.4 增加循环对年龄程序的改进  
while True:  
 age = input("请输入您的年龄(退出请输入exit)：")  
 ageMax = 100  
 if age == "exit":  
 break  
 else:  
 print("你的年龄是：" + age)  
 intAge = int(age) # 使用intAge变量将三次类型转换减少到一次  
 if (intAge < 1) or (intAge > ageMax):  
 print("年龄输入错误！不能小于1岁或大于100岁")  
 else:  
 afterYears = input("请输入过去的年份：")  
 ageResult = intAge + int(afterYears)  
 if ageResult > ageMax:  
 print("过去的年份输入错误！不能大于100岁")  
 else:  
 sentence = "过去" + afterYears + "年后，" + "你的年龄是" + str(ageResult) + "岁。"  
 print(sentence)

### 1.2.3 控制语句-030循环for

# 1.1for循环语句和range()函数实现指定次数的循环  
# for关键字；  
# 一个变量名；  
# in关键字；  
# 调用range()方法，最多传入3个参数；  
# 冒号；  
# 从下一行开始，缩退的代码块（称为for子句）。  
for value in range(5):  
 print("value的值是：" + str(value))  
  
# 等价的while循环  
value = 0  
while value < 5:  
 print("value的值是：" + str(value))  
 value = value + 1  
  
# 1.2 range()函数可以有三个参数.  
# 前两个参数分别是起始值和终止值，第三个参数是“步长”。  
# 步长是每次迭代后循环变量增加的值。  
for value in range(2, 5):  
 print("value的值是：" + str(value))  
  
for value in range(100, 200, 2):  
 print("value的值是：" + str(value))  
  
for value in range(200, 100, -2):  
 print("value的值是：" + str(value))

# 2 基础部分

## 2.1 list

### 2.1.1 list(列表定义)

# 字符串列表的定义  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10'] #字符串列表  
print(peoples)  
  
# “列表”是一个值，它包含多个字构成的序列。  
# 术语“列表值”指的是列表本身（它作为一个值，可以保存在变量中，或传递给函数，像所有其他值一样），而不是指列表值之内的那些值。  
# 列表值看起来像这样：['people01', 'people02', 'people03', 'people04']。  
# 就像字符串值用引号来标记字符串的起止一样，列表用左方括号开始，右方括号结束，即[]。  
# 列表中的值也称为“表项”。表项用逗号分隔（就是说，它们是“逗号分隔的”）。  
  
  
# 数值列表的定义  
ages = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
print(ages)

### 2.1.2 list(列表的遍历)

# 列表的遍历  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
for people in peoples: # 遍历列表  
 print(people)  
  
# 更友好的遍历输出  
for people in peoples: # 遍历列表  
 print('目前遍历到的元素值是:'+people)  
print("列表的元素总数是:" + str(len(peoples)) + '个')

### 2.1.3 list(列表的访问)

# 列表使用索引值访问  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
# 假定列表['people01', 'people02', 'people03', 'people04']保存在名为peoples的变量中。  
# Python代码peoples[0]将求值为'people01'，peoples[1]将求值为'people02'，依此类推。  
# 列表后面方括号内的整数被称为“下标”。列表中第一个值的下标是0，第二个值的下标是1，第三个值的下标是2，依此类推。  
print(peoples[0])  
print(peoples[3])  
print(peoples[3].title()) # 首字母大写  
print(peoples[-1]) # 取列表最后一个元素  
message = '这次选中的是05号：'+peoples[4] # 元素运算  
print(message)  
  
# 列表使用切片访问  
print(peoples[0:3]) # 使用切片访问  
print(peoples[:3]) # 使用切片访问，默认从头开始  
print(peoples[2:]) # 使用切片访问，默认到达列表尾端  
print(peoples[-4:]) # 使用切片访问，获取列表最后几个  
  
# 遍历使用切片的列表  
for people in peoples[-6:]:   
 print(people)  
  
# index()方法用列表值来找到下标。  
# 如果列表中存在重复的值，就返回它第一次出现的下标。  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peopleValue = 'people05'  
peopleIndex = peoples.index(peopleValue)  
print("用户" + peopleValue + "在列表片中的下标为：" + str(peopleIndex))  
  
# peopleValue = 'people11'  
# peopleIndex = peoples.index(peopleValue)  
# print(peopleIndex)  
  
# 对列表值的检查  
# 利用in和not in操作符，可以确定一个值否在列表中。  
# 像其他操作符一样，in和not in用在表达式中，连接两个值：一个要在列表中查找的值，以及待查找的列表。  
# 这些表达式将求值为布尔值。  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
judgePeople = 'people01' in peoples  
print(judgePeople)  
judgePeople = 'people01' not in peoples  
print(judgePeople)  
judgePeople = 'people11' in peoples  
print(judgePeople)  
  
# 列表对变量的多重赋值  
# 多重赋值技巧是一种快捷方式，让你在一行代码中，用列表中的值为多个变量赋值。  
# 变量的数目和列表的长度必须严格相等。  
  
# 传统赋值语句  
peoples = ['people01', 'people02', 'people03']  
peopleOne = peoples[0]  
peopleTwo = peoples[1]  
peopleThree = peoples[2]  
print(peopleOne, peopleTwo, peopleThree)  
  
# 多重赋值  
peopleOne, peopleTwo, peopleThree = peoples  
print(peopleOne, peopleTwo, peopleThree)

### 2.1.4 list(列表元素的增删改查)

# 列表元素的修改  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoples[0] = 'people001' # 修改元素值  
print(peoples)  
  
# 列表元素的添加  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoples.append('people11') # 添加元素  
print(peoples)  
  
# 空列表添加元素  
peoples = [] # 空列表  
print(peoples)  
peoples.append('people01') # 为空列表添加元素  
print(peoples)  
  
# 在指定位置插入列表元素  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoples.insert(1, 'people01to02') # 在指定索引插入元素  
print(peoples)  
  
# 删除指定位置的列表元素  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
del peoples[9] # 删除指定索引的元素  
del peoples[-1]  
print(peoples)  
  
# 删除指定值的列表元素  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoples.remove('people05') # 删除指定值的元素(第一个指定的值)  
print(peoples)  
  
# 使用栈的方式访问并删除列表  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peopleDelete = peoples.pop() # 弹出最后一个元素，栈的概念  
print(peopleDelete)  
print(peoples)  
  
# 增加列表对年龄连续输入的存储  
ageList = []  
while True:  
 age = input("请输入您的年龄(退出请输入exit)：")  
 ageMax = 100  
 if age == "exit":  
 break  
 else:  
 print("你的年龄是：" + age)  
 intAge = int(age) # 使用intAge变量将三次类型转换减少到一次  
 if (intAge < 1) or (intAge > ageMax):  
 print("年龄输入错误！不能小于1岁或大于100岁")  
 else:  
 ageList.append(intAge)  
 print("已经输入的年龄列表为：")  
 print(ageList)

### 2.1.5 list(列表的排序)

# 列表的永久排序和逆排序  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
print(peoples)  
peoples.sort() # 列表的永久排序，升序  
print(peoples)  
peoples.sort(reverse = True) # 列表的永久逆排序，降序  
print(peoples)  
  
# 列表的临时排序和逆排序  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
print(sorted(peoples)) # 列表的临时排序，升序  
print(peoples)  
print(sorted(peoples, reverse = True)) # 列表的临时排序，降序  
print(peoples)  
  
# 列表的逆序  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
peoples.reverse() # 列表逆序  
print(peoples)

### 2.1.6 list(列表长度)

# 列表长度  
peoples = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peopleNumber = len(peoples) # 获取列表的长度，元素个数  
print(peopleNumber)  
  
# 空列表的长度  
peoples = []  
peopleNumber = len(peoples) # 获取列表的长度，元素个数  
print(peopleNumber)

### 2.1.7 list(数值列表)

# range的含义  
for value in range(1, 10):   
 print(value)  
  
for value in range(10):   
 print(value)  
  
# 使用range创建列表  
numbers = list(range(1, 10))   
print(numbers)  
for nuimber in numbers:  
 print(nuimber)  
  
# 使用range创建列表,带步长  
numbers = list(range(1, 10, 2))   
print(numbers)  
  
numbers = list(range(2, 11, 2))   
print(numbers)  
  
# 使用值计算建立列表  
numbers = []  
for value in range(1, 11):   
 numbers.append(value\*\*2)  
print(numbers)  
  
# 列表的统计计算  
numbers = list(range(1, 11))  
print(min(numbers))   
print(max(numbers))  
print(sum(numbers))  
print(sum(numbers)/len(numbers))

### 2.1.8 list(列表推导)

# 使用循环构建数值列表1-100  
numbers = []  
for value in range(1, 101):  
 numbers.append(value)  
print(numbers)  
  
# 列表推导式构建数值列表1-100  
numbers = [number for number in range(1, 101)] # 列表的构造器，切记使用[]一旦使用()就成了生成器  
print(numbers)  
  
# 使用循环构建偶数数值列表1-100  
evenNumbers = []  
for value in range(1, 101):  
 if value % 2 == 0:  
 evenNumbers.append(value)  
print(evenNumbers)  
  
# 带条件的列表推导式构建偶数数值列表1-100  
evenNumbers = [number for number in range(1, 101) if number % 2 == 0] # 带条件的列表的构造器，输出1-100内的偶数  
print(evenNumbers)  
# 表达式 for 变量 in 列表 if 条件  
  
# 带条件的列表推导式案例  
oddNumbers = [number for number in range(1, 101) if number % 2 != 0] # 带条件的列表的构造器，输出1-100内的偶数  
print(oddNumbers)  
evenNumbers = [number for number in range(1, 101) if number % 2 == 0 if number > 49] # 带多个条件的列表推导式  
print(evenNumbers)  
  
# 使用循环对列表值进行计算  
numbers = []  
for value in range(1,6):  
 numbers.append(value)  
numberCal = []  
for number in numbers:  
 numberCal.append(number\*\*2)  
print(numberCal)  
  
# 带计算的列表推导式对列表值进行计算  
numbers = [value\*\*2 for value in range(1, 6)]   
# 使用值计算建立列表,更精简的方法,6行代码缩减为一行  
# 这种写法除了更加高效之外，也更加简短，涉及的语法元素也更少。  
# 在大型程序中，这意味着更少的错误，代码也更容易阅读和理解。  
print(numbers)  
# 表达式 for 变量 in 列表  
  
# 使用多个列表的推导式  
names = ['高等数学', '计算机网络基础', '大学英语', 'Python编程基础', 'Linux操作系统基础']  
scores = ['一班', '二班', '三班']  
nameScores = [(name, score) for name in names for score in scores]  
print(nameScores)

### 2.1.9 list(列表的拼接)

# 不使用列表拼接的传统做法  
listOne = [1, 2, 3, 4, 5]  
listTwo = [6, 7, 8, 9, 10]  
for value in listTwo:  
 listOne.append(value)  
print(listOne)  
  
# 使用+号进行列表拼接  
listOne = [1, 2, 3, 4, 5]  
listTwo = [6, 7, 8, 9, 10]  
listOne = listOne + listTwo  
print(listOne)

### 2.1.10 list(字符串列表与字符串操作)

# 1.1 字符串拼接的低效做法  
# 拼接任意不可变序列都会生成一个新的序列对象。  
# Python字符串是不可变的。  
# 由于不变性，字符串可以作为字典的键或set的元素，因为一旦初始化之后字符串的值就不会改变；  
# 另一方面，每当需要修改过的字符串时（即使只是微小的修改），都需要创建一个全新的字符串实例。  
namesString = ""  
namesList = ['周杰伦', '李健', '谢霆锋', '庾澄庆']  
for name in namesList:  
 namesString = namesString + name  
print(namesString)  
  
# 1.2 字符串拼接的高效方法join  
# join()方法速度更快（对于大型列表来说更是如此），并不意味着在所有需要拼接两个字符串的情况下都应该使用这一方法。  
# 虽然这是一种广为认可的做法，但并不会提高代码的可读性。可读性是很重要的！  
namesList = ['周杰伦', '李健', '谢霆锋', '庾澄庆']  
namesString = "".join(namesList)  
print(namesString)  
  
# 1.3 使用特定字符作为join字符串连接分隔符  
namesList = ['周杰伦', '李健', '谢霆锋', '庾澄庆']  
namesString = "-".join(namesList)  
print(namesString)  
  
namesList = ['周杰伦', '李健', '谢霆锋', '庾澄庆']  
namesString = ",".join(namesList)  
print(namesString)  
  
# 1.4 使用split()将字符串拆分成字符串列表  
words = "My name is Simon"  
wordsList = words.split()  
print(wordsList)  
  
words = '''你好,  
吃饭了没？  
没吃的话一起去吃饭？  
我在教学楼的。'''  
wordsList = words.split()  
print(wordsList)  
wordsList = words.split('\n')  
print(wordsList)  
  
dateTime = "2019-01-8-14-33"  
wordsList = dateTime.split('-')  
print(wordsList)

### 2.1.11 tempCodeRunnerFile

numbers = [value\*\*2 for value in range(1, 6)] # 使用值计算建立列表,更精简的方法  
print(numbers)

## 2.2 set

### 2.2.1 new

listOne = ['常州', '常州', '盐城', '徐州', '苏州', '苏州']  
listTwo = ['常州', '南京', '台州', '苏州', '苏州']  
  
# listCross = []  
# for one in listOne:  
# if one in listTwo:  
# listCross.append(one)  
# print(listCross)  
# setCross = set(listCross)  
# listCross = list(setCross)  
# print(listCross)  
  
# listAll = listTwo  
# for one in listOne:  
# if one not in listTwo:  
# listAll.append(one)  
# print(listAll)  
# setAll = set(listAll)  
# listAll = list(setAll)  
# print(listAll)  
   
setOne = set(listOne)  
setTwo = set(listTwo)  
setCross = setOne & setTwo  
setAll = setOne | setTwo  
setOneUnique = setOne - setTwo  
setTwoUnique = setTwo - setOne  
  
print(setCross)  
print(setAll)  
print(setOneUnique)  
print(setTwoUnique)

### 2.2.2 set(集合定义)

# 集合是许多唯一对象的聚集  
# 集合是一种鲁棒性很好的数据结构，当元素顺序的重要性不如元素的唯一性和测试元素是否包含在集合中的效率时，大部分情况下这种数据结构是很有用的。  
listOne = [1, 1, 2, 3, 3, 4]  
print(set(listOne))  
listTwo = list(set(listOne))  
print(listTwo)

### 2.2.3 set(集合运算)

# 集合的交集---取两个集合的共有部分  
setOne = set(list(range(1, 8)))  
setTwo = set(list(range(5, 15)))  
print(setOne)  
print(setTwo)  
setThree = setOne & setTwo  
print(setThree)  
  
# 集合的并集--取两个集合所有  
setOne = set(list(range(1, 8)))  
setTwo = set(list(range(5, 15)))  
print(setOne)  
print(setTwo)  
setThree = setOne | setTwo  
print(setThree)  
  
# 集合的差集--取一个集合与另一个集合的不同部分  
setOne = set(list(range(1, 8)))  
setTwo = set(list(range(5, 15)))  
print(setOne)  
print(setTwo)  
setThree = setOne-setTwo  
print(setThree)  
setThree = setTwo-setOne  
print(setThree)  
  
# 生源地的例子  
listOne = ['常州', '常州', '盐城', '徐州', '苏州', '苏州']  
listTwo = ['常州', '南京', '台州', '苏州', '苏州']  
setOne = set(listOne)  
setTwo = set(listTwo)  
  
setCross = setOne & setTwo  
print(setCross)  
  
setAll = setOne | setTwo  
print(setAll)

## 2.3 dictionary

### 2.3.1 dictionary(字典定义)

# 字典定义  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
print(people)

### 2.3.2 dictionary(字典的遍历)

# 字典键值对的遍历  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
for key,value in people.items():  
 print("键：" + key)  
 print("值：" + str(value))  
  
# 字典键的遍历  
for key in people.keys(): #只遍历键  
 print("键：" + key)  
  
# 字典值的遍历  
for key in people.values(): #只遍历值  
 print("值：" + str(value))

### 2.3.3 dictionary(字典的访问)

# 访问字典值  
people = {'身高': '170cm', '体重': '60kg', '年龄': 45}  
print(people['身高'])  
print('这个人是：' + str(people['年龄']) + '岁。')  
  
# 检查字典中是否存在键或值  
# in和not in操作符可以检查值是否存在于列表中。  
# 也可以利用这些操作符，检查某个键或值是否存在于字典中。  
people = {'身高': '170cm', '体重': '60kg', '年龄': 45}  
key = '身高'  
judgeResult = key in people.keys()  
print(judgeResult)  
value = '60kg'  
judgeResult = value in people.keys()  
print(judgeResult)  
  
# get()方法使用键来获取值  
# get()方法，它有两个参数：要取得其值的键，以及如果该键不存在时，返回的备用值。  
people = {'身高': '170cm', '体重': '60kg', '年龄': 45}  
key = '身高'  
getResult = people.get(key)  
print(getResult)  
  
key = '籍贯'  
getResult = people.get(key, '没有键值')  
print(getResult)

### 2.3.4 dictionary(字典的增删改查)

# 字典键值对的增加  
people = {'身高': '170cm', '体重': '60kg', '年龄': 45}  
people['籍贯'] = '江苏'  
people['性别'] = '男'  
print(people)  
  
# 字典值的修改  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
people['年龄'] = 40  
print(people)  
  
# 字典键值对的删除  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
del people['身高']  
print(people)  
  
# 字典键值对的查找  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
age = people.get('年龄')  
print(age)  
  
# # 字典构建年龄程序的应用  
# ageNameDict = {}  
# while True:  
# name = input("请输入您的姓名(退出请输入exit)：")  
# age = input("请输入您的年龄(退出请输入exit)：")  
# if age == "exit" or name == "exit":  
# break  
# else:  
# ageNameDict[name] = age  
# print(ageNameDict)  
# # 程序的主要问题输名字退出exit无法直接退出，还属输完年龄退出才能终止程序  
  
  
# # 字典构建年龄程序的第一次迭代  
# ageNameDict = {}  
# loopSig = True # 引入了循环运行的标志位变量  
# while loopSig:  
# name = input("请输入您的姓名(退出请输入exit)：")  
# if name == "exit":  
# loopSig = False  
# continue  
# else:  
# age = input("请输入您的年龄(退出请输入exit)：")  
# if age == "exit":  
# loopSig = False  
# continue  
# else:  
# ageNameDict[name] = age  
# print(ageNameDict)  
# # 要对年龄有一个判断并支持重新输入  
  
# 字典构建年龄程序的第二次迭代  
ageNameDict = {}  
loopSig = True # 引入了循环运行的标志位变量  
ageMax = 100  
while loopSig:  
 name = input("请输入您的姓名(退出请输入exit)：")  
 if name == "exit":  
 loopSig = False  
 continue  
 else:   
 while loopSig:  
 age = input("请输入您的年龄(必须小于100，退出请输入exit)：")  
 if age == "exit":  
 loopSig = False  
 continue  
 else:  
 intAge = int(age)  
 if intAge > ageMax:  
 continue  
 else:  
 ageNameDict[name] = intAge  
 print(ageNameDict)  
 break

### 2.3.5 dictionary(字典的排序)

# 字典排序  
people = {'身高1':160,'身高2':170,'身高3':170,'身高4':140}  
for height in sorted(people.values()):  
 print('身高是：' + str(height))

### 2.3.6 dictionary(使用列表推导式构建字典)

#使用列表推导式构建字典  
names = ['高等数学','计算机网络基础','大学英语','Python编程基础','Linux操作系统基础']  
scores = [78,85,87,90,76]  
nameScoreDic = dict([(name,score) for name in names for score in scores]) #生成方式有问题  
print(nameScoreDic)  
nameScoreSet = [(name,score) for name in names for score in scores]  
print(nameScoreSet)  
  
#使用列表推导式构建字典  
names = ['高等数学','计算机网络基础','大学英语','Python编程基础','Linux操作系统基础']  
scores = ['一班','二班','三班','四班','五班']  
# scores = [78,85,87,90,76]  
nameScores = dict([(name,score) for name in names for score in scores]) #生成方式有问题  
print(nameScores)  
  
#使用列表推导式快速更换键值  
nameClassDic = {'高等数学': '一班', '计算机网络基础': '二班', '大学英语': '三班', 'Python编程基础': '四班', 'Linux操作系统基础': '五班'}  
classNameDic = {v:k for k,v in nameClassDic.items()}  
print(classNameDic)  
  
#使用列表推导式构建单词和长度的字典  
words = ['python','java','c#','sql']  
wordsLengthDic = {word : len(word) for word in words}  
print(wordsLengthDic)

### 2.3.7 tempCodeRunnerFile

# 字典键值对的增加  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
people['籍贯'] = '江苏'  
people['性别'] = '男'  
print(people)  
  
# 字典值的修改  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
people['年龄'] = 40  
print(people)  
  
# 字典键值对的删除  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
del people['身高']  
print(people)  
  
# 字典键值对的查找  
people = {'身高':'170cm','体重':'60kg','年龄':45}  
age = people.get('年龄')  
print(age)

## 2.4 tuple

### 2.4.1 tuple(元组定义)

# 定义元组  
tuples = (1,2,3,4,5,6,7,8,9,10)  
print(tuples)

### 2.4.2 tuple(元组的遍历)

# 元组的遍历  
tuples = (1,2,3,4,5,6,7,8,9,10)  
for oneTuple in tuples:  
 print(oneTuple)

### 2.4.3 tuple(元组的访问)

# 访问元组  
tuples = ('people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
print(tuples[0])

### 2.4.4 tuple(元组的不可编辑)

# 元组的修改  
tuples = (1,2,3,4,5,6,7,8,9,10)  
tuples[0] = 3 #元组元素无法赋值，只能整个定义

## 2.5 while

### 2.5.1 while(循环的运行)

# while 循环  
show = "\n请输入文本消息，程序将重复显示"  
show += "\n输入'exit'将退出程序"  
message = ""  
while message != 'exit':  
 message = input(show)  
 if message != 'exit':  
 print(message)  
  
#while循环使用状态控制跳出  
show = "\n请输入文本消息，程序将重复显示"  
show += "\n输入'exit'将退出程序"  
active = True  
while active:  
 message = input(show)  
 if message == 'exit':  
 active = False #使用while循环内的状态位进行控制  
 else:  
 print(message)

### 2.5.2 while(限定次数的循环)

# while 限定次数的循环  
count = 1  
while count<=5:  
 print('这是循环的第' + str(count) + '次')  
 count += 1  
  
count = 1  
active = True  
while active:  
 print('这是循环的第' + str(count) + '次')  
 count += 1  
 if count == 6:  
 active = False #使用while循环内的状态位进行控制  
  
count = 1  
active = True  
while active:  
 print('这是循环的第' + str(count) + '次')  
 if count == 5:  
 active = False #使用while循环内的状态位进行控制  
 count += 1  
  
for value in range(1,6): #使用for循环  
 print('这是循环的第' + str(value) + '次')

### 2.5.3 while(循环处理列表)

# while循环处理列表  
#for 循环是一种遍历列表的有效方式， 但在for 循环中不应修改列表，  
# 否则将导致Python难以跟踪其中的元素。  
# 要在遍历列表的同时对其进行修改，可使用while 循环。  
# 通过将while 循环同列表和字典结合起来使用，可收集、存储并组织大量输入，供以后查看和显示。  
peoplesForCheck = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoplesChecked = []  
while peoplesForCheck:  
 people = peoplesForCheck.pop()  
 print('目前在检查的用户是:' + people.title())  
 peoplesChecked.append(people)  
print(peoplesChecked)

### 2.5.4 while(continue和break)

# while的continue  
peoplesForCheck = ['people01', 'people02', 'women03', 'people04', 'people05', 'women06', 'people07', 'people08','people09', 'people10']  
womenChecked = []  
while peoplesForCheck:  
 people = peoplesForCheck.pop()  
 if people[:6] == 'people':  
 continue #直接跳到循环体开始，不再执行循环体中continue后面的语句  
 # else:  
 print('目前在检查的女性用户是:' + people.title())  
 womenChecked.append(people)  
print(womenChecked)  
  
  
#1.2 while的break  
password = 'pass'  
while True:  
 passwordInput = input('请输入密码:')  
 if passwordInput == password:  
 print('密码正确。')  
 break  
print('你已成功进入系统')

## 2.6 if-else

### 2.6.1 ifelse(条件检查)

# 基本的状态检查  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
for people in sorted(peoples):  
 if people == 'people02'.lower(): #等于  
 print(people.upper())  
 else:  
 print(people.title())  
  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
for people in sorted(peoples):  
 if people != 'people02'.lower(): #不等于  
 print(people.upper())  
 else:  
 print(people.title())

### 2.6.2 ifelse(多重检查)

# 多重判定  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
print(peoples[0][6:8]) #正向截取  
print(peoples[0][6:]) #正向截取  
print(peoples[0][-2:]) #反向截取  
for people in sorted(peoples):  
 if int(people[6:8]) >=7: #数字判定  
 print(people.upper())  
 elif int(people[6:8]) <= 3:  
 print(people.title())  
 else:  
 print(people.lower())

### 2.6.3 ifelse(多条件检查)

# 多条件判定  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
for people in peoples:  
 if people == 'people02' or people == 'people01': #多个条件的判定  
 print('你选中的是01或者02的用户')  
 elif people == 'people10':  
 print('你选中的是10的用户') #取消了else的默认操作

### 2.6.4 ifelse(列表的值检查)

# 列表的值判定  
peoples = ['people10', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people01']  
if 'people02' in peoples:  
 print("'people02'在列表中") #'和"的用法  
if 'people11' not in peoples:  
 print('"people11"不在列表中') #'和"的用法  
  
# 空列表的判定  
peoples = []  
if peoples: #列表为空的判定  
 print('列表有数据')  
else:  
 print('列表为空')

### 2.6.5 ifelse(多列表检查)

# 多列表判定  
peoplesOne = ['people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10']  
peoplesTwo = ['people01', 'people02', 'people03', 'people04', 'people05']  
  
for peopleOne in peoplesOne:  
 if peopleOne in peoplesTwo:  
 print(peopleOne+'在两个列表中都存在')  
 else:  
 print(peopleOne+'仅在peopleOne列表中存在')

## 2.7 function

### 2.7.1 calculateGeometryFunc

# 函数作为返回值计算几何特性  
def aeraRectangle(length,width): #矩形面积  
 return length\*width  
  
def volumeCuboid(length,width,height): #长方体体积  
 return length\*width\*height  
  
def calculateGeometry(calculateFor):  
 if calculateFor == 'aerajuxing':  
 return aeraRectangle #函数作为返回值  
 if calculateFor == 'volumejuxing':  
 return volumeCuboid #函数作为返回值  
  
# calGeometry = calculateGeometry('aerajuxing')  
# print(str(calGeometry(4,5))  
# calGeometry = calculateGeometry('volumejuxing')  
# print(calGeometry(4,5,6))

### 2.7.2 calculateGeometryFuncUse

import calculateGeometryFunc as calGeo  
calGeometry = calGeo.calculateGeometry('aerajuxing')  
print(str(calGeometry(4,5)))  
calGeometry = calGeo.calculateGeometry('volumejuxing')  
print(calGeometry(4,5,6))  
  
  
  
  
# from calculateGeometryFunc import calculateGeometry as calGeo  
# calGeometry = calGeo('aerajuxing')  
# print(str(calGeometry(4,5)))  
# calGeometry = calGeo('volumejuxing')  
# print(calGeometry(4,5,6))

### 2.7.3 function(函数定义)

# 基本的函数定义  
def printName(userName):  
 print('打印名字为' + userName + '的用户')  
  
printName('张三')  
  
def sumTwo(one,two):  
 print('输入的两个数的和是' + str(one + two))  
  
sumTwo(1,2)  
sumTwo(1.1,2.1)  
sumTwo('123','456')

### 2.7.4 function(形参和实参)

# 位置实参的顺序  
def printInfo(name,gender):  
 print('姓名为' + name + '的用户性别是' + gender + '。')  
  
printInfo('张三','男')  
printInfo('男','张三') #实参顺序错误  
printInfo(gender = '男',name = '张三') #关键字实参  
  
# 形参的默认值  
def printUserInfo(name,gender = '男'): #形参的默认值  
 print('姓名为' + name + '的用户性别是' + gender + '。')  
  
printUserInfo('张三')  
printUserInfo('李丽','女')

### 2.7.5 function(形参的默认值)

# 形参的默认值  
def printUserInfo(name,gender = '男'): #形参的默认值  
 print('姓名为' + name + '的用户性别是' + gender + '。')  
  
printUserInfo('张三')  
printUserInfo('李丽','女')

### 2.7.6 function(函数的返回值)

# 函数返回值  
def get\_fullName(lastname,firstname):  
 return (lastname+firstname)  
  
fullName = get\_fullName('张','三')  
print(fullName)  
  
def get\_name(lastname,firstname,middleName = ''):#使用默认值增强灵活性  
 return(lastname + middleName + firstname)  
  
fullName = get\_name('张', '三')  
print(fullName)  
fullName = get\_name('张', '三', '贵')  
print(fullName)  
  
# 函数可以返回多个值  
def get\_name(lastname,firstname,middleName = ''): # 使用默认值增强灵活性  
 return(lastname, middleName + firstname)  
lastname, firstname = get\_name('张', '三', '贵')  
print("输入的姓是：" + lastname + "，输入的名是：" + firstname + "。")

### 2.7.7 function(局部变量和全局变量)

#1.1局部变量和全局变量的错误示例  
def judge(value):  
 if value == "全局变量":  
 localValue = "全局变量"#处于局部作用域的变量，被称为“局部变量”。  
 else:  
 localValue = "局部变量"  
 return localValue  
   
globalValue = "全局变量" #处于全局作用域的变量，被称为“全局变量”。  
result = judge(globalValue)  
print(result)  
# print(localValue) #局部变量不能在全局作用域内使用  
#一个变量必是其中一种，不能既是局部的又是全局的。  
#在被调用函数内赋值的变元和变量，处于该函数的“局部作用域”。  
  
#1.2 global语句  
def judge(value):  
 global localValue  
 if value == "全局变量":  
 localValue = "全局变量"#处于局部作用域的变量，被称为“局部变量”。  
 else:  
 localValue = "局部变量"  
 return localValue  
globalValue = "全局变量" #处于全局作用域的变量，被称为“全局变量”。  
result = judge(globalValue)  
print(result)  
print(localValue)  
#如果需要在一个函数内修改全局变量，就使用global语句。  
#如果在函数的顶部有global localValue这样的代码，它就告诉Python，  
#在这个函数中，localValue指的是全局变量，所以不要用这个名字创建一个局部变量  
  
# 有4条法则，来区分一个变量是处于局部作用域还是全局作用域：  
# 1．如果变量在全局作用域中使用（即在所有函数之外），它就总是全局变量。  
# 2．如果在一个函数中，有针对该变量的global语句，它就是全局变量。  
# 3．否则，如果该变量用于函数中的赋值语句，它就是局部变量。  
# 4．但是，如果该变量没有用在赋值语句中，它就是全局变量。

### 2.7.8 function(任意数量的实参形式)

# 任意数量的实参-元祖  
def print\_peoples(\*peoples):  
 for people in peoples:  
 print(people)  
print\_peoples('people01')  
print\_peoples('people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
  
##单个参数与多个参数的混合使用  
# def print\_info(gender,\*peoples):  
# print('性别是' + gender + '性的同学如下：')  
# for people in peoples:  
# print(people)  
# print\_info('女','people01', 'people02', 'people03')  
# print\_info('男','people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')

### 2.7.9 function(任意数量的关键字实参)

# 任意数量的关键字实参-字典  
def build\_info(\*\*user):  
 userInfo = {}  
 for key,value in user.items():  
 userInfo[key] = value  
 return userInfo  
userInfo = build\_info(身高 = '170cm',体重 = '60kg',年龄 = 45)  
print(userInfo)

### 2.7.10 function(函数的Import)

# 函数的import  
import function\_60\_forImport  
  
# 函数包使用  
function\_60\_forImport.printName('张三')  
function\_60\_forImport.sumTwo(1,2)  
function\_60\_forImport.sumTwo(1.1,2.1)  
function\_60\_forImport.sumTwo('123','456')  
  
# 函数位置实参的顺序  
function\_60\_forImport.printInfo('张三','男')  
function\_60\_forImport.printInfo('男','张三') #实参顺序错误  
function\_60\_forImport.printInfo(gender = '男',name = '张三') #关键字实参  
  
# 函数形参的默认值  
function\_60\_forImport.printUserInfo('张三')  
function\_60\_forImport.printUserInfo('李丽','女')  
  
# 函数返回值  
fullName = function\_60\_forImport.get\_fullName('张','三')  
print(fullName)  
  
fullName = function\_60\_forImport.get\_name('张','三')  
print(fullName)  
  
fullName = function\_60\_forImport.get\_name('张','三','贵')  
print(fullName)  
  
# 函数任意数量的实参-元祖  
function\_60\_forImport.print\_peoples('people01')  
function\_60\_forImport.print\_peoples('people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
function\_60\_forImport.print\_info('女','people01', 'people02', 'people03')  
function\_60\_forImport.print\_info('男','people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
# 函数任意数量的关键字实参-字典  
userInfo = function\_60\_forImport.build\_info(身高 = '170cm',体重 = '60kg',年龄 = 45)  
print(userInfo)

### 2.7.11 function(导入模块的别名)

from function\_60\_forImport import build\_info  
from function\_60\_forImport import print\_info as printPeoplesInfo #导入函数使用别名  
  
printPeoplesInfo('女','people01', 'people02', 'people03')  
printPeoplesInfo('男','people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
userInfo = build\_info(身高 = '170cm',体重 = '60kg',年龄 = 45)  
print(userInfo)

### 2.7.12 function(别名的使用)

import function\_60\_forImport as peopleFunction #导入模块使用别名  
  
# 1.1 基本的函数定义  
peopleFunction.printName('张三')  
  
peopleFunction.sumTwo(1,2)  
peopleFunction.sumTwo(1.1,2.1)  
peopleFunction.sumTwo('123','456')  
  
# 1.2 位置实参的顺序  
peopleFunction.printInfo('张三','男')  
peopleFunction.printInfo('男','张三') #实参顺序错误  
peopleFunction.printInfo(gender = '男',name = '张三') #关键字实参  
  
# 1.3 形参的默认值  
peopleFunction.printUserInfo('张三')  
peopleFunction.printUserInfo('李丽','女')  
  
# 1.4 返回值  
fullName = peopleFunction.get\_fullName('张','三')  
print(fullName)  
  
  
fullName = peopleFunction.get\_name('张','三')  
print(fullName)  
fullName = peopleFunction.get\_name('张','三','贵')  
print(fullName)  
  
# 1.5 任意数量的实参-元祖  
peopleFunction.print\_peoples('people01')  
peopleFunction.print\_peoples('people01', 'people02', 'people03', 'people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
  
peopleFunction.print\_info('女','people01', 'people02', 'people03')  
peopleFunction.print\_info('男','people04', 'people05', 'people06', 'people07', 'people08','people09', 'people10')  
  
# 1.6 任意数量的关键字实参-字典  
userInfo = peopleFunction.build\_info(身高 = '170cm',体重 = '60kg',年龄 = 45)  
print(userInfo)

### 2.7.13 function(函数作为参数)

#一个没有拆分的代码范例  
def calculateGeometry(calName,para1,para2 = 0,para3 = 0):  
 if(calName == 'aeraRectangle'):  
 return para1\*para2  
 if(calName == 'volumeCuboid'):  
 return para1\*para2\*para3  
 if(calName == 'aeraCircle'):  
 return math.pi \* para1\*\*2  
 if(calName == 'volumeSphere'):  
 return 4\*math.pi \* para1\*\*3/3  
  
  
# # 函数作为参数计算几何特性  
# import math #引入数学计算包  
# def aeraRectangle(length,width): #矩形面积  
# return length \* width  
# # aeraRectangleResult = aeraRectangle(3,4)  
# # aeraOutput = str(aeraRectangleResult)  
# # print(aeraOutput)  
  
# def volumeCuboid(length,width,height): #长方体体积  
# return length \* width \* height  
  
# # def aeraCircle(radius): #圆形面积  
# # return math.pi \* radius\*\*2  
  
# def volumeSphere(radius): #球形体积  
# return 4\*math.pi \* radius\*\*3/3  
  
# def aeraCircle(radius): #圆形面积  
# return math.pi \* radius\*\*2  
  
  
  
  
  
  
  
  
  
  
  
  
# 函数作为参数计算几何特性  
import math #引入数学计算包  
def aeraRectangle(length,width,parameterC = 0): #矩形面积  
 return length \* width  
  
def volumeCuboid(length,width,height): #长方体体积  
 return length \* width \* height  
  
def aeraCircle(radius,parameterB=0,parameterC = 0): #圆形面积  
 return math.pi \* radius\*\*2  
  
def volumeSphere(radius,parameterB=0,parameterC = 0): #球形体积  
 return 4\*math.pi \* radius\*\*3/3  
  
def calculateGeometry(calFunc,parameterA,parameterB = 0,parameterC = 0): #使用函数作为参数，并使用了默认参数值  
 return calFunc(parameterA,parameterB,parameterC)  
  
print(calculateGeometry(aeraRectangle,4,5))  
print(calculateGeometry(volumeCuboid,1,2,3))  
print(calculateGeometry(aeraCircle,0.5))  
print(calculateGeometry(volumeSphere,1))

### 2.7.14 function(函数作为返回值)

# 函数作为返回值计算几何特性  
def aeraRectangle(length,width): #矩形面积  
 return length\*width  
  
def volumeCuboid(length,width,height): #长方体体积  
 return length\*width\*height  
  
def calculateGeometry(calculateFor):  
 if calculateFor == 'aerajuxing':  
 return aeraRectangle #函数作为返回值  
 if calculateFor == 'volumejuxing':  
 return volumeCuboid #函数作为返回值  
  
calGeometry = calculateGeometry('aerajuxing')  
print(str(calGeometry(4,5)))  
calGeometry = calculateGeometry('volumejuxing')  
print(calGeometry(4,5,6))

### 2.7.15 function(函数的闭包)

# 函数的闭包，创建形式相似的函数  
# Y = aX+b  
def functionDef(a,b,x): #三个参数  
 return a\*x+b  
print(str(functionDef(2,5,5)))  
  
def functionUpDef(a,b):  
 def functionRun(x): #一个函数和它的环境变量合在一起，就构成了一个闭包（Closure）。a，b就是函数functionRun的环境变量  
 return a\*x+b  
 return functionRun  
  
funcRun = functionUpDef(2,5)  
print(str(funcRun(5)))  
print(str(funcRun(6)))

### 2.7.16 function\_60\_forImport

# 1.1 基本的函数定义  
def printName(userName):  
 print('打印名字为' + userName + '的用户')  
  
def sumTwo(one,two):  
 print('输入的两个数的和是' + str(one + two))  
  
# 1.2 位置实参的顺序  
def printInfo(name,gender):  
 print('姓名为' + name + '的用户性别是' + gender + '。')  
  
# 1.3 形参的默认值  
def printUserInfo(name,gender = '男'): #形参的默认值  
 print('姓名为' + name + '的用户性别是' + gender + '。')  
  
# 1.4 返回值  
def get\_fullName(firstName,lastName):  
 return (firstName+lastName)  
  
def get\_name(firstName,lastName,middleName = ''):#使用默认值增强灵活性  
 return(firstName + middleName + lastName)  
  
# 1.5 任意数量的实参-元祖  
def print\_peoples(\*peoples):  
 for people in peoples:  
 print(people)  
  
def print\_info(gender,\*peoples):  
 print('性别是' + gender + '性的同学如下：')  
 for people in peoples:  
 print(people)  
  
# 1.6 任意数量的关键字实参-字典  
def build\_info(\*\*user):  
 userInfo = {}  
 for key,value in user.items():  
 userInfo[key] = value  
 return userInfo

## 2.8 class

### 2.8.1 class(类的基本定义)

# # 1.1 类的基本定义和引用  
# class People():  
# def \_\_init\_\_(self,objectName,objectAge):  
# self.name = objectName #类的属性定义  
# self.age = objectAge  
# def study(self): #类的方法定义  
# print(self.name + '年龄是' + str(self.age) + '岁正在学习中。。。')  
  
  
# people = People('王小强',32) #对象(实例)的初始化  
# people01 = People('张小花',24)  
  
# message = people.name + '和' +people01.name + '一起去吃饭'  
# print(message)  
  
# people.study()  
# people01.study()  
  
# 1.1 类的基本定义和引用  
class People():  
 def \_\_init\_\_(self,objectName,objectAge,objectGender):  
 self.name = objectName #类的属性定义  
 self.age = objectAge  
 self.gender = objectGender  
 def study(self): #类的方法定义  
 print(self.name + '年龄是' + str(self.age) + '岁正在学习中。。。')  
 def ageAdd(self): #方法对属性的操作  
 self.age = self.age + 1  
  
  
people = People('王小强',32,'男') #对象(实例)的初始化  
people01 = People('张小花',24,'女')  
  
print(people.name + '的性别是：' + people.gender + ',年龄是：' + str(people.age) + '。')  
print(people01.name + '的性别是：' + people01.gender + ',年龄是：' + str(people01.age) + '。')  
  
people.ageAdd()  
print(people.name + '的性别是：' + people.gender + ',年龄是：' + str(people.age) + '。')  
print(people01.name + '的性别是：' + people01.gender + ',年龄是：' + str(people01.age) + '。')  
  
  
  
  
  
  
  
  
# people01.name = '李晓红' #对象的属性改变  
# people01.study()  
# print( 'name属性的值：' + people01.name)

### 2.8.2 class(类属型的默认值)

# 1.2 类属型的默认值  
class People():  
 def \_\_init\_\_(self,name,age):  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = '女' #类的属性默认值  
 def study(self): #类的方法定义  
 print(self.name + '年龄是' + str(self.age) + '岁' +'性别是' + self.gender+'正在学习中。。。')  
  
people = People('王小强',32) #对象(实例)的初始化  
people.gender = '男'   
people.study()  
print( 'name属性的值：' + people.name)  
  
people01 = People('张小花',24)  
people01.study()  
print( 'name属性的值：' + people01.name)  
  
class People():  
 def \_\_init\_\_(self,name,age,gender = '女'):#类的属性默认值  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = gender  
 def study(self): #类的方法定义  
 print(self.name + '年龄是' + str(self.age) + '岁' +'性别是' + self.gender+'正在学习中。。。')  
  
people = People('王小强',32,'男') #对象(实例)的初始化  
# people.gender = '男'   
people.study()  
print( 'name属性的值：' + people.name)  
  
people01 = People('张小花',24)  
people01.study()  
print( 'name属性的值：' + people01.name)

### 2.8.3 class(通过方法改变属性值)

# 1.3 通过方法修改属性值  
class People():  
 def \_\_init\_\_(self,name,age):  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = '女' #类的属性默认值  
 def study(self): #类的方法定义  
 print(self.name + '年龄是' + str(self.age) + '岁' +'性别是' + self.gender+'正在学习中。。。')  
 def setMale(self):  
 self.gender = '男'  
  
people = People('王小强',32) #对象(实例)的初始化  
people.setMale()  
people.study()  
print( 'name属性的值：' + people.name)

### 2.8.4 class(类的继承)

# 1.4 类的继承  
class People():  
 def \_\_init\_\_(self,name,age):  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = '女' #类的属性默认值  
 def getPeopleInfo(self): #类的方法定义  
 print(self.name + '，年龄是' + str(self.age) + '岁，' +'性别是' + self.gender+'。')  
 def setMale(self):  
 self.gender = '男'  
  
class Student(People): #继承自People  
 def \_\_init\_\_(self,name,age):  
 super().\_\_init\_\_(name,age)  
 self.school = '常州信息职业技术学院' #子类新增的属性  
 def getSchoolInfo(self): #子类新增的函数  
 print(self.name + '来自学校：' + self.school)  
 def setSchool(self, schoolName): #子类新增的函数  
 self.school = schoolName  
  
student = Student('李莉',22)  
student.getPeopleInfo()  
student.getSchoolInfo()  
student.setSchool('常州工学院')  
student.getSchoolInfo()

### 2.8.5 class(子类的属性和方法定义)

# 1.5子类的属性和方法定义  
class People():  
 def \_\_init\_\_(self,name,age):  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = '女' #类的属性默认值  
 def getPeopleInfo(self): #类的方法定义  
 print(self.name + '，年龄是' + str(self.age) + '岁，' +'性别是' + self.gender+'。')  
 def setMale(self):  
 self.gender = '男'  
  
class Student(People):  
 def \_\_init\_\_(self,name,age,grade):#增加子类的属性  
 super().\_\_init\_\_(name,age)  
 self.grade = grade #增加子类的属性  
 def getPeopleInfo(self): #重写父类的方法  
 print(self.name + '，年龄是' + str(self.age) + '岁，' +'性别是' + self.gender+'。'  
 + '现在上' + str(self.grade) + '年级。')  
  
student = Student('李莉',22,2)  
student.getPeopleInfo()

### 2.8.6 class(类的实例用作属性)

# 1.6 类的实例用作属性  
class People():  
 def \_\_init\_\_(self,name,age):  
 self.name = name #类的属性定义  
 self.age = age  
 self.gender = '女' #类的属性默认值  
 def getPeopleInfo(self): #类的方法定义  
 print(self.name + '，年龄是' + str(self.age) + '岁，' +'性别是' + self.gender+'。')  
 def setMale(self):  
 self.gender = '男'  
  
class Student(People):  
 def \_\_init\_\_(self,name,age,grade,motherName,motherAge):#增加子类的属性  
 super().\_\_init\_\_(name,age)  
 self.grade = grade #增加子类的属性  
 self.mother = People(motherName,motherAge)  
 def getPeopleInfo(self): #重写父类的方法  
 print(self.name + '，年龄是' + str(self.age) + '岁，' +'性别是' + self.gender+'。'  
 + '现在上' + str(self.grade) + '年级。')  
  
student = Student('李莉',22,2,'张丽',48)  
student.mother.getPeopleInfo()  
student.getPeopleInfo()

## 2.9 综合案例

### 2.9.1 010基本字典查案例的迭代

#本案例主要目标为根据输入的名字，通过主键关联来找到对应的成绩  
  
#查找的基本算法  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
nameInput = input('please input name:')  
for key,name in keyNameDict.items():  
 if name == nameInput: #用户输入的名字与keyNameDict中的值相等则执行下面的代码  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[key ])) #将keyNameDict的键直接用作keyScoreDict的键来获取对应的值  
  
#增加针对查找不到数据的处理  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
keyFind = 0 #使用该标志位来代表是否根据name找到key  
nameInput = input('please input name:')  
for key,name in keyNameDict.items():  
 if name == nameInput:  
 keyFind = key #找到key则使用key来替换标志位  
if keyFind == 0: #根据标志位的变化来决定输出  
 print('no find')  
else:  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind]))  
  
#增加用户连续输入提高易用性的处理  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
Active = True #使用该标志位来进行程序的循环  
while Active:  
 keyFind = 0  
 nameInput = input('please input name:')  
 for key,name in keyNameDict.items():  
 if name == nameInput:  
 keyFind = key   
 if keyFind == 0:  
 print('no find')  
 else:  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind]))  
  
  
#根据输入条件结束程序  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
Active = True  
while Active:  
 keyFind = 0  
 nameInput = input('please input name:')  
 if nameInput == 'exit': #使用特定的输入来改变循环的执行条件  
 Active = False  
 else:  
 for key,name in keyNameDict.items():  
 if name == nameInput:  
 keyFind = key   
 if keyFind == 0:  
 print('no find')  
 else:  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind]))  
  
#一旦查找到姓名对应的键值，处理后就跳出循环  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
Active = True  
while Active:  
 keyFind = 0  
 nameInput = input('please input name:')  
 if nameInput == 'exit': #使用特定的输入来改变循环的执行条件  
 Active = False  
 else:  
 for key,name in keyNameDict.items():  
 if name == nameInput:  
 keyFind = key   
 break #一旦查找到主键，处理后使用break跳出循环  
 if keyFind == 0:  
 print('no find')  
 else:  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind]))

### 2.9.2 020使用函数封装的字典查询

#使用函数封装相应的查找和输出功能  
import findScore  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
  
Active = True  
while Active:  
 nameInput = input('please input name:')  
 if nameInput == 'exit':  
 Active = False  
 else:  
 keyFind = findScore.findKey(keyNameDict,nameInput)  
 result = findScore.findScore(keyScoreDict,keyFind)  
 findScore.findResult(nameInput,result)  
  
#使用对函数再次进行封装的函数  
import findScoreAll  
keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
Active = True  
while Active:  
 nameInput = input('please input name:')  
 if nameInput == 'exit':  
 Active = False  
 else:  
 findScoreAll.findScoreAll(keyNameDict,keyScoreDict,nameInput)  
 findScoreAll.findScoreSame(keyNameDict,keyScoreDict,nameInput)

### 2.9.3 findScore

#该函数封装了使用名字查找学号的功能  
def findKey(keyNameDict,nameInput):  
 for key,name in keyNameDict.items():  
 if name == nameInput:  
 return key  
 return 0  
  
#该函数封装了使用学号找到成绩的功能  
def findScore(keyScoreDict,keyFind):  
 if keyFind == 0:  
 return -1  
 else:  
 return keyScoreDict[keyFind]  
  
#该函数使用用户输入的名字和查找到的成绩形成输出的功能  
def findResult(nameInput,result):  
 if result == -1:  
 print('no find')  
 else:  
 print("find "+ nameInput +"'s " +"score is "+str(result))

### 2.9.4 findScoreAll

import findScore  
  
def findScoreAll(keyNameDict,keyScoreDict,nameInput):  
 keyFind = findScore.findKey(keyNameDict,nameInput)  
 result = findScore.findScore(keyScoreDict,keyFind)  
 findScore.findResult(nameInput,result)  
  
def findScoreSame(keyNameDict,keyScoreDict,nameInput):  
 findScore.findResult(nameInput,findScore.findScore(keyScoreDict,findScore.findKey(keyNameDict,nameInput)))

### 2.9.5 new

keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
keyScoreDict = {10001:71,10002:77,10003:99}  
  
active =True  
while active:  
 nameInput = input('please input name:')  
 if nameInput == 'exit':  
 active = False  
 else:  
 keyFind = 0  
 for key,name in keyNameDict.items():  
 if name == nameInput: #用户输入的名字与keyNameDict中的值相等则执行下面的代码  
 keyFind = key  
 if keyFind == 0:  
 print('no find!')  
 else:  
 print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind])) #将keyNameDict的键直接用作keyScoreDict的键来获取对应的值  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
# keyNameDict = {10001:'zhangsan',10002:'lisi',10003:'wangwu'}  
# keyScoreDict = {10001:71,10002:77,10003:99}  
  
# keyFind = 0 #使用该标志位来代表是否根据name找到key  
# nameInput = input('please input name:')  
# for key,name in keyNameDict.items():  
# if name == nameInput:  
# keyFind = key #找到key则使用key来替换标志位  
# if keyFind == 0: #根据标志位的变化来决定输出  
# print('no find')  
# else:  
# print("find "+ nameInput +"'s " +"score is "+str(keyScoreDict[keyFind]))

# 3 进阶部分

## 3.1 exception

### 3.1.1 exception(异常的发生和处理)

# 1.1 异常的发生  
numerator = float(input('请输入分子：')) #输入非数字型值会引发异常  
denumerator = float(input('请输入分母：')) #输入非数字型值会引发异常，0也会引发异常  
result = numerator/denumerator  
print('分子' + str(numerator) + '分母' + str(denumerator) + '的结果为：' + str(result) )  
  
# 1.2 异常的基本处理  
try:  
 numerator = float(input('请输入分子：')) #输入非数字型值会引发异常  
 denumerator = float(input('请输入分母：')) #输入非数字型值会引发异常，0也会引发异常  
 result = numerator/denumerator  
except ValueError: #输入非数字型值会引发异常  
 print('输入的数据不是数字型引发异常！！！')  
except ZeroDivisionError: #分母输入0引发异常  
 print('分母为零引发异常！！！')  
else:  
 print('分子' + str(numerator) + '分母' + str(denumerator) + '的结果为：' + str(result) )

### 3.1.2 exception(一个完整的计算器)

# 1.1 一个完整的除法计算器  
while True:  
 print('-----除法计算器，输入"q"就会停止计算。-----')  
 numeratorInput = input('请输入分子：')  
 if numeratorInput == 'q':  
 break #使用break跳出循环体  
 denumeratorInput = input('请输入分母：')  
 if denumeratorInput == 'q':  
 break  
 try:  
 numerator = float(numeratorInput) #输入非数字型值会引发异常  
 denumerator = float(denumeratorInput) #输入非数字型值会引发异常，0也会引发异常  
 result = numerator/denumerator  
 except ValueError: #输入非数字型值会引发异常  
 print('输入的数据不是数字型引发异常！！！')  
 except ZeroDivisionError: #分母输入0引发异常  
 print('分母为零引发异常！！！')  
 else:  
 print('分子' + str(numerator) + '分母' + str(denumerator) + '的结果为：' + str(result) )  
 # print('分子{0}分母{1}的结果为：{2}'.format(str(numerator),str(denumerator),str(result))) #使用format函数占位显示  
 # print('分子' + '{:g}'.format(numerator) + '分母' + '{:g}'.format(denumerator) + '的结果为：' + '{:g}'.format(result)) #使用format函数去除小数点后面的0和不必要的.  
  
  
# 1.2 一个完整的除法计算器[抓取所有异常]  
while True:  
 print('-----除法计算器，输入"q"就会停止计算。-----')  
 numeratorInput = input('请输入分子：')  
 if numeratorInput == 'q':  
 break #使用break跳出循环体  
 denumeratorInput = input('请输入分母：')  
 if denumeratorInput == 'q':  
 break  
 try:  
 numerator = float(numeratorInput) #输入非数字型值会引发异常  
 denumerator = float(denumeratorInput) #输入非数字型值会引发异常，0也会引发异常  
 result = numerator/denumerator  
 except Exception:  
 print('请检查输入的是否是数字或分母是否为零！')  
 else:  
 print('分子' + str(numerator) + '分母' + str(denumerator) + '的结果为：' + str(result) )

### 3.1.3 exception(综合案例)

#一个原始的除法器  
numberOne = input('请输入分子：')  
numberTwo = float(input('请输入分母：'))  
result = numberOne/numberTwo  
print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
  
#带异常检测的除法器  
try:  
 numberOne = float(input('请输入分子：'))  
 numberTwo = float(input('请输入分母：'))   
 result = numberOne/numberTwo  
except:  
 print('用户输入错误!')  
else:  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
  
#可以连续输入带异常检测的除法器  
active = True  
while active:  
 try:  
 numberOne = input('请输入分子：')  
 if numberOne == 'exit':  
 break  
 else:  
 numberOneFloat = float(numberOne)  
 numberTwo = input('请输入分母：')  
 if numberTwo == 'exit':  
 break  
 else:  
 numberTwoFloat = float(numberTwo)  
 result = numberOneFloat/numberTwoFloat  
 except:  
 print('用户输入错误!')  
 else:  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
  
  
#可以连续输入带异常检测的除法器  
active = True  
numberOne = ''  
numberTwo = ''  
exitSig = 'exit'  
while active:  
 if numberOne == exitSig or numberTwo == exitSig:  
 break  
 else:  
 while True:  
 numberOne = input('请输入分子：')  
 if numberOne == exitSig:  
 break  
 else:  
 try:  
 numberOneFloat = float(numberOne)  
 except ValueError:  
 print('输入数据类型不是数字')  
 else:   
 break   
 while True and numberOne != exitSig:  
 numberTwo = input('请输入分母：')  
 if numberTwo == exitSig:  
 break  
 else:  
 try:  
 numberTwoFloat = float(numberTwo)  
 result = numberOneFloat/numberTwoFloat  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
 except ValueError:  
 print('输入数据类型不是数字')  
 except ZeroDivisionError:  
 print('分母不能为零！')  
 else:  
 break  
   
  
#可以连续输入带异常检测的除法器,改进的代码,去掉了一个if判断和一个循环  
active = True  
numberOne = ''  
numberTwo = ''  
exitSig = 'exit'  
while active:   
 numberOne = input('请输入分子：')  
 if numberOne == exitSig:  
 active = False  
 else:  
 try:  
 numberOneFloat = float(numberOne)  
 except ValueError:  
 print('输入数据类型不是数字')  
 continue  
 else:   
 while active :  
 numberTwo = input('请输入分母：')  
 if numberTwo == exitSig:  
 active = False  
 else:  
 try:  
 numberTwoFloat = float(numberTwo)  
 result = numberOneFloat/numberTwoFloat  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
 except ValueError:  
 print('输入数据类型不是数字')  
 except ZeroDivisionError:  
 print('分母不能为零！')  
 else:  
 break

### 3.1.4 new

# #可以连续输入带异常检测的除法器  
# exitSig = 'exit'  
# while True:  
# try:  
# #-----------------------------------  
# numberOne = input('请输入分子：')  
# if numberOne == exitSig:  
# break  
# else:  
# numberOneFloat = float(numberOne)  
# #------------------------------------  
# numberTwo = input('请输入分母：')  
# if numberTwo == exitSig:  
# break  
# else:  
# numberTwoFloat = float(numberTwo)  
# #------------------------------------  
# result = numberOneFloat/numberTwoFloat  
# except:  
# print('用户输入错误!')  
# else:  
# print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
  
  
  
  
active = True  
numberOne = ''  
numberTwo = ''  
exitSig = 'exit'  
while active:   
 numberOne = input('请输入分子：')  
 if numberOne == exitSig:  
 break  
 else:  
 try:  
 numberOneFloat = float(numberOne)  
 except ValueError:  
 print('输入数据类型不是数字')  
 continue  
 else:   
 while active :  
 numberTwo = input('请输入分母：')  
 if numberTwo == exitSig:  
 active = False  
 else:  
 try:  
 numberTwoFloat = float(numberTwo)  
 result = numberOneFloat/numberTwoFloat  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
 except ValueError:  
 print('输入数据类型不是数字')  
 except ZeroDivisionError:  
 print('分母不能为零！')  
 else:  
 break

### 3.1.5 tempCodeRunnerFile

#可以连续输入带异常检测的除法器,改进的代码  
active = True  
numberOne = ''  
numberTwo = ''  
exitSig = 'exit'  
while active:   
 numberOne = input('请输入分子：')  
 if numberOne == exitSig:  
 active = False  
 else:  
 try:  
 numberOneFloat = float(numberOne)  
 except ValueError:  
 print('输入数据类型不是数字')  
 continue  
 else:   
 while active :  
 numberTwo = input('请输入分母：')  
 if numberTwo == exitSig:  
 active = False  
 else:  
 try:  
 numberTwoFloat = float(numberTwo)  
 result = numberOneFloat/numberTwoFloat  
 print('分子 ' + str(numberOne) + '除以分母 '+ str(numberTwo) + '的结果为:' + str(result))  
 except ValueError:  
 print('输入数据类型不是数字')  
 except ZeroDivisionError:  
 print('分母不能为零！')  
 else:  
 break

## 3.2 fileRead

### 3.2.1 fileread(文件的打开和读取)

# 1.1 文件的打开和读取  
# -\*- coding:utf-8 -\*-  
with open('D:\\PythonStudy-V1.0\\020 upper\\010 fileRead\\fileRead.txt',encoding='UTF-8') as fileObject: #转义符'\\'和读取中文的编码'UTF-8',linux系统使用斜杠'/'  
 contents = fileObject.read()  
 print(contents)

### 3.2.2 fileRead(绝对路径和相对路径)

# 1.1 相对路径  
with open('./pythonStudy/020进阶部分/010fileRead/fileRead.txt',encoding='UTF-8') as fileObject: #./代表相对路径中的工作目录  
 contents = fileObject.read()  
 print(contents)  
  
# 1.2 父文件夹表达  
with open('../PythonStudy-V1.0/pythonStudy/020进阶部分/010fileRead/fileRead.txt',encoding='UTF-8') as fileObject: #../代表相对路径中的工作目录的上一级目录  
 contents = fileObject.read()  
 print(contents)  
  
# 1.3 当前文件的获取  
import os  
filePath = os.path.abspath(\_\_file\_\_) #当前文件的绝对路径  
print(filePath)  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
print(fileFolderPath)

### 3.2.3 fileread(去除空行)

# 1.2 文件读取去除最后一个空行  
with open('./pythonStudy/020进阶部分/010fileRead/fileRead.txt',encoding='UTF-8') as fileObject: #转义符'\\'和读取中文的编码'UTF-8'  
 contents = fileObject.read()  
 print(contents.rstrip()) #因为read() 到达文件末尾时返回一个空字符串，而将这个空字符串显示出来时就是一个空行。rstrip() 删除 string 字符串末尾的指定字符（默认为空格）.

### 3.2.4 fileread(文件的逐行读取和预处理)

# 1.3 文件的逐行读取和内容预处理  
fileFullPath = './pythonStudy/020进阶部分/010fileRead/fileRead.txt'   
lineNumber = 1  
lineLength = 0   
with open(fileFullPath,encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1

### 3.2.5 fileread(文件内容存入列表)

# 1.4 文件的内容存入列表  
fileFullPath = './pythonStudy/020进阶部分/010fileRead/fileRead.txt'  
lineNumber = 1  
lineLength = 0   
with open(fileFullPath,encoding='UTF-8') as fileObject:  
 lines = fileObject.readlines()  
for line in lines:  
 print(line.strip())

### 3.2.6 fileread(基本的文件内容查找)

# 1.5 基本的文件内容查找  
fileFullPath = './pythonStudy/020进阶部分/010fileRead/fileRead.txt'  
lineNumber = 1  
lineLength = 0   
lines = []  
with open(fileFullPath,encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 lines.append(lineProcessed)  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1  
charFind = input('请输入你要查找的字符：')  
lineNumber = 0  
for line in lines:  
 charCount = line.count(charFind)  
 lineNumber +=1   
 print('第' + str(lineNumber) + '段找到' + "'" + charFind + "'" + '共计' + str(charCount) + '个')

### 3.2.7 fileread(文件内容查找的改进版本)

# 1.5 基本的文件内容查找，结合while的改进版本  
fileFullPath = './pythonStudy/020进阶部分/010fileRead/fileRead.txt'  
lineNumber = 1  
lineLength = 0   
lines = []  
with open(fileFullPath,encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 lines.append(lineProcessed)  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1  
  
active = True  
while active:  
 charFind = input('请输入你要查找的字符(输入"quit"结束分析)：')  
 if charFind != 'quit':  
 lineNumber = 0  
 charCountSum = 0  
 for line in lines:  
 charCount = line.count(charFind) #计算每一行找到的字符个数  
 charCountSum += charCount #计算找到字符的合计数  
 lineNumber +=1   
 print('第' + str(lineNumber) + '行找到' + "'" + charFind + "'" + '共计' + str(charCount) + '个')  
 print('找到' + "'" + charFind + "'" + '共计' + str(charCountSum) + '个')   
 else:  
 active = False

## 3.3 fileWrite

### 3.3.1 filewrite(写入文件)

# 1.1 写入文件  
fileFullPath = './pythonStudy/020进阶部分/020fileWrite/fileWrite.txt'   
#（'w' ）告诉Python，我们要以写入模式 打开这个文件。  
# 打开文件时，可指定读取模式 （'r' ）、写入模式 （'w' ）、附加模式 （'a' ）或让你能够读取和写入文件的模式（'r+' ）。  
# 写入（'w' ）模式打开文件时千万要小心，因为如果指定的文件已经存在，Python将在返回文件对象前清空该文件。  
# 如果你要写入的文件不存在，函数open() 将自动创建它  
with open(fileFullPath,'w',encoding='UTF-8') as fileObject:   
 fileObject.write('我写入了一行数据')

### 3.3.2 filewrite(多行写入文件)

# 1.2 写入多行数据  
fileFullPath = './pythonStudy/020进阶部分/fileWrite.txt'   
with open(fileFullPath,'w',encoding='UTF-8') as fileObject: #（'w' ）告诉Python，我们要以写入模式 打开这个文件。  
 fileObject.write('我写入了一行数据。\n') #使用换行转义符'\n'  
 fileObject.write('我又写入了一行数据。\n') #使用换行转义符'\n'  
 fileObject.write('我又又写入了一行数据。\n') #使用换行转义符'\n'

### 3.3.3 filewrite(追加模式写入文件)

# 1.3 附加模式写入文件，不清空  
fileFullPath = './pythonStudy/020进阶部分/fileWrite.txt'   
with open(fileFullPath,'a',encoding='UTF-8') as fileObject: #（'a' ）告诉Python，我们要以附加模式 打开这个文件。  
 fileObject.write('附加：我写入了一行数据。\n') #使用换行转义符'\n'  
 fileObject.write('附加：我又写入了一行数据。\n') #使用换行转义符'\n'  
 fileObject.write('附加：我又又写入了一行数据。\n') #使用换行转义符'\n'  
  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject: #（'r' ）告诉Python，我们要以只读模式 打开这个文件。  
 print(fileObject.read())

## 3.4 wordsAnalysis

### 3.4.1 wordsAnalysis(分词处理)

# 1.1 将内容拆分成单词列表的程序  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 words = contents.split()  
print(words)  
print(len(words))

### 3.4.2 wordsAnalysis(分词清洗)

# 1.2 分词的基本清洗  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
wordsProcessed = []  
wordsUnique = []  
wordsCount = {}  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 words = contents.split()  
 for word in words:  
 wordProcessed = word.strip()  
 wordProcessed = wordProcessed.replace('.','')  
 wordProcessed = wordProcessed.replace(',','')  
 wordProcessed = wordProcessed.replace('"','')  
 wordProcessed = wordProcessed.replace(':','')  
 wordProcessed = wordProcessed.replace('”','')  
 wordProcessed = wordProcessed.replace('“','')  
 wordProcessed = wordProcessed.replace('?','')  
 wordProcessed = wordProcessed.replace(';','')  
 wordProcessed = wordProcessed.lower()  
 # wordProcessed = word.strip().replace('.','').replace(',','').replace('"','').replace(':','').replace('”','').replace('“','').replace('?','').replace(';','').lower()  
 wordsProcessed.append(wordProcessed)  
print(words)  
print(len(words))

### 3.4.3 wordsAnalysis(分词的词频计算)

# 1.3 基本单词列表词频计算  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
wordsProcessed = []  
wordsUnique = []  
wordsCount = {}  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 words = contents.split()  
 for word in words:  
 wordProcessed = word.strip()  
 wordProcessed = wordProcessed.replace('.','')  
 wordProcessed = wordProcessed.replace(',','')  
 wordProcessed = wordProcessed.replace('"','')  
 wordProcessed = wordProcessed.replace(':','')  
 wordProcessed = wordProcessed.replace('”','')  
 wordProcessed = wordProcessed.replace('“','')  
 wordProcessed = wordProcessed.replace('?','')  
 wordProcessed = wordProcessed.replace(';','')  
 wordProcessed = wordProcessed.lower()  
 # wordProcessed = word.strip().replace('.','').replace(',','').replace('"','').replace(':','').replace('”','').replace('“','').replace('?','').replace(';','').lower()  
 wordsProcessed.append(wordProcessed)  
for wordProcessed in wordsProcessed:  
 if wordProcessed not in wordsUnique:  
 wordsUnique.append(wordProcessed)  
for wordUnique in wordsUnique:  
 wordCount = wordsProcessed.count(wordUnique)  
 wordsCount[wordUnique] = wordCount  
# for word,count in wordsCount.items():  
# 如果写作key=lambda item:item[0]的话则是选取第一个元素作为比较对象  
# 也就是key值作为比较对象。lambda x:y中x表示输出参数，y表示lambda 函数的返回值  
for word,count in sorted(wordsCount.items(),key = lambda item:item[1]):  
 print('单词"{0}"出现在文章中的次数为{1}次。'.format(word,str(count)))  
  
print(len(wordsProcessed))  
print(len(wordsUnique))

### 3.4.4 wordsAnalysis(分词的正则匹配处理)

# 1.4 重新看分词的匹配  
import re  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
wordRE = re.compile(r"([^a-z,^A-Z]{1,3})") #将分词后的特殊字符拿出来建立列表  
wordsAbnomal = []  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 words = contents.split()  
 for word in words:  
 matchObject = re.match(wordRE,word)  
 if matchObject:  
 if matchObject.group(1) not in wordsAbnomal:  
 wordsAbnomal.append(matchObject.group(1))  
print(wordsAbnomal)  
  
# 改进的re分词处理  
import re  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
wordRE = re.compile(r"[^a-z]\*([a-z]{1,20})")   
wordsCleaned = []  
with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 words = contents.split()  
 for word in words:  
 matchObject = re.match(wordRE,word)  
 if matchObject:  
 if matchObject.group(1) not in wordsCleaned:  
 wordsCleaned.append(matchObject.group(1))  
print(wordsCleaned)

### 3.4.5 wordsAnalysis(模块封装)

# 1.5 使用封装函数进行处理  
import wordsAnalysisFunction as wordsProcess  
fileFullPath = './pythonStudy/020进阶部分/030wordsAnalysis/wordsAnalysis.txt'  
  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,1)  
print(wordsCount)  
  
for word,count in wordsCount.items():  
 print('单词"{0}"出现在文章中的次数为{1}次。'.format(word,str(count)))  
  
  
import os  
import csv  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/fileAnalysisUpper.csv' #按照csv格式进行写入，csv是标准数据处理格式  
with open(fileFullPath,'w',newline = '') as csvFile:  
 csvWrite = csv.writer(csvFile)  
 for row in wordsCount.items():  
 csvWrite.writerow(row)

### 3.4.6 wordsAnalysisFunction

import re #导入正则表达式处理模块re  
def wordMatch(wordForClean): #定义单个分词的清洗  
 # wordRE = re.compile(r"[‘,“,(]\*([a-z]{1,20})")  
 wordRE = re.compile(r"[^a-z]\*([a-z]{1,20})")  
 matchObject = re.match(wordRE,wordForClean)  
 return matchObject  
  
def wordsClean(textForDivide): #定义单词列表的清洗  
 wordsForClean = textForDivide.split()  
 wordsCleaned = []  
 for wordForClean in wordsForClean:  
 matchObject = wordMatch(wordForClean.lower()) #全部转成小写的处理  
 if matchObject:  
 wordsCleaned.append(matchObject.group(1))  
 return wordsCleaned  
  
def textDivide(fileFullPath): #定义拆分单词的函数  
 wordsCleaned = []  
 with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 wordsCleaned = wordsClean(contents)  
 return wordsCleaned  
   
def analyzeWordsCount (wordsCleaned,minCount): #定义基本的统计函数  
 wordsUnique = []  
 wordsCount = {}  
 for word in wordsCleaned:  
 if word not in wordsUnique:  
 wordsUnique.append(word)  
 for wordUnique in wordsUnique:  
 wordCount = wordsCleaned.count(wordUnique)  
 if wordCount >= minCount: #此处利用参数来选择最小的词频  
 wordsCount[wordUnique] = wordCount  
 wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #使用该排序将字典变成有序的元组  
 wordsCountDict = dict(wordsCountOrder) #将有序的元组转换为字典  
 return wordsCountDict #返回字典

### 3.4.7 wordsProcess

def fileRead(fileName):   
 '''  
 本函数将打开与本文件同一文件夹路径下的文件进行读取  
 传入参数为文件的全名[string]  
 传出参数为读取的文件内容[string]  
 '''  
 import os  
 fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
 fileFullPath = fileFolderDir + fileName #使用字符串拼接，得到需要打开文件的绝对路径  
 with open(fileFullPath,encoding = 'utf-8') as fileObject: #使用绝对路径打开文件，并将文件作为一个对象fileObject  
 fileContent = fileObject.read() #读取文件对象的内容，并赋值给变量fileContent  
 return fileContent  
  
def fileClean(fileContent):  
 '''  
 本函数将读取的文件内容形成清洗后的单词列表  
 传入参数为读取的文件内容[string]  
 传出的内容为单词列表[list]  
 '''  
 #将标点符号替换为空格  
 fileContent = fileContent.replace('?',' ')  
 fileContent = fileContent.replace(';',' ')  
 fileContent = fileContent.replace('.',' ')  
 fileContent = fileContent.replace(',',' ')  
 fileContent = fileContent.replace(':',' ')  
 fileContent = fileContent.replace('"',' ')  
 fileContent = fileContent.replace("'",' ')  
 fileContent = fileContent.replace('(',' ')  
 fileContent = fileContent.replace(')',' ')  
 fileContent = fileContent.replace("’",' ')  
 fileContent = fileContent.replace("“",' ')  
 fileContent = fileContent.replace("”",' ')  
 wordsCleaned = fileContent.lower().split() #split方法使用空格作为分隔符将字符串fileContent拆分成列表  
 for word in wordsCleaned:  
 if len(word) == 1 and word != "i": #去除单词长度为1，且不是大写I的单词  
 wordsCleaned.remove(word)  
 return wordsCleaned  
  
def fileWordsUnique(wordsCleaned):  
 '''  
 本函数将从单词列表获得单词列表中出现的单词  
 传入参数为读取的单词列表[list]  
 传出的内容为单词列表中出现的不重复单词列表[list]  
 '''  
 #第一代获取不重复单词列表的处理代码  
 wordsUnique = [] #初始化空列表wordsUnique，用于存储words列表中出现的不重复单词  
 for word in wordsCleaned:  
 if wordsUnique.count(word) == 0: #如果在wordsUnique列表中找不到单词word就加入该列表  
 wordsUnique.append(word)  
  
 # #第二代获取不重复单词列表的处理代码  
 # wordsUniqueSet = set(wordsCleaned)  
 # wordsUnique = list(wordsUniqueSet)  
 return wordsUnique  
  
def fileWordsCount(wordsCleaned,wordsUnique):  
 '''  
 本函数将单词列表中的单词进行计数  
 传入参数为读取的单词列表wordsCleaned[list]，不重复单词列表wordsUnique[list]  
 传出的内容为单词及计数的字典[dict]  
 '''  
 wordsCount = {}  
 for word in wordsUnique:  
 wordsCount[word] = wordsCleaned.count(word) #将words列表中出现word的次数作为值，word作为键加入字典wordsCount  
 wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #对字典的value进行排序  
 wordsCountDict = dict(wordsCountOrder) #将元祖转换为字典  
 return wordsCountDict  
  
def fileWordsCountAll(fileName):  
 fileContent = fileRead(fileName)  
 wordsCleaned = fileClean(fileContent)  
 wordsUnique = fileWordsUnique(wordsCleaned)  
 wordsCountDict = fileWordsCount(wordsCleaned,wordsUnique)  
 return wordsCountDict

### 3.4.8 wordsProcessUse

import wordsProcess  
fileName = "\\wordsAnalysis.txt"  
# fileContent = wordsProcess.fileRead(fileName)  
# wordsCleaned = wordsProcess.fileClean(fileContent)  
# wordsUnique = wordsProcess.fileWordsUnique(wordsCleaned)  
# wordsCountDict = wordsProcess.fileWordsCount(wordsCleaned,wordsUnique)  
wordsCountDict = wordsProcess.fileWordsCountAll(fileName)  
print(wordsCountDict)  
  
# help(wordsProcess.fileRead)  
  
# from pyecharts import Bar #引入数据可视化模块pyechart中的Bar直方图  
# bar\_x = []  
# bar\_y = []  
# for x,y in wordsCountDict.items():#遍历已排序的字典wordsCountDict的键、值  
# if y > 4: #将词频出现5次及以上的单词进行显示  
# bar\_x.append(x)  
# bar\_y.append(y)  
# bar = Bar('words count','English words count')  
# bar.use\_theme('dark')  
# bar.add('文章1词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
# bar.add('文章2词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
# bar.add('文章3词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
# bar.render('wordsCount.html')  
  
# from pyecharts import WordCloud  
# wordCloud = WordCloud(width = 900,height = 600)  
# wordCloud.add('英文单词词频',bar\_x,bar\_y,word\_size\_range = [60,180])  
# wordCloud.render('wordsCloud.html')

### 3.4.9 wordsResultWrite

import wordsProcess  
import os  
import csv  
   
fileName = "\\wordsAnalysis.txt"  
wordsCountDict = wordsProcess.fileWordsCountAll(fileName)  
  
fileNameToWrite = '\\wordsResult.csv'  
fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderDir + fileNameToWrite #使用字符串拼接，得到需要写入文件的绝对路径  
with open(fileFullPath,'w',newline = '',encoding = 'utf-8') as csvObject:  
 csvWriter = csv.writer(csvObject)  
 for wordCount in wordsCountDict.items():  
 csvWriter.writerow(wordCount)

## 3.5 lamdaExpression

### 3.5.1 lamdaExpression(lambda表达式初步)

#1.1 lamda 表达式初步  
#定义了加法函数lambda x, y: x+y。并将其赋值给变量add  
# 这样变量add便成为具有加法功能的函数。  
  
# lambda表达式，通常是在需要一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数。  
# lambda所表示的匿名函数的内容应该是很简单的，如果复杂的话，干脆就重新定义一个函数了，使用lambda就有点过于执拗了。  
# lambda就是用来定义一个匿名函数的，如果还要给他绑定一个名字的话，就会显得有点画蛇添足，通常是直接使用lambda函数。  
  
addFunction = lambda x,y:x+y #":"前是输入，后是输出  
print(addFunction(10,20))

### 3.5.2 lamdaExpression(lambda表达式filter)

#1.2 lamda 表达式对列表的处理，filter  
number = [x for x in range(1,101)] #列表的推导式  
evenNumber = list(filter(lambda x : x%2 == 0 ,number)) #使用lambda获取原有列表中偶数  
print(evenNumber)  
filterNumber = list(filter(lambda x : x > 50 , number)) #使用lambda获取原有列表中大于50的数  
print(filterNumber)  
  
# 其实Python的for..in..if语法已经很强大，并且在易读上胜过了lambda。  
number = [x for x in range(1,101)] #列表的推导式  
oddNumber = list(filter(lambda x : x%2 != 0 ,number)) #使用filter获取100以内的奇数的例子  
print(oddNumber)  
oddNumber = [x for x in number if x % 2 != 0]  
print(oddNumber)

### 3.5.3 lamdaExpression(lambda表达式map)

# lamda 表达式对列表的处理，map  
number = [x for x in range(1,101)] #列表的推导式  
doubleNumber = list(map(lambda x : x\*2 , number)) #使用lambda对原有列表中"每一个"元素进行处理  
print(doubleNumber)  
numberTwo = [x for x in range(101,201)] #列表的推导式  
addNumberList = list(map(lambda x,y: x+y , number, numberTwo)) #使用lambda对元素数量相同的两个列表中"每一个"元素进行处理  
print(addNumberList)  
  
  
# lamda 表达式也可以用列表推导式来完成同样的功能  
doubleNumber = [x\*2 for x in range(1,101)] #列表推导式的代码更简洁 doubleNumber = list(map(lambda x : x\*2 , number))  
print(doubleNumber)  
  
#列表推导式不同的地方  
doubleNumber = [x\*2 for x in range(1,101)] #列表推导式的代码更简洁 doubleNumber = list(map(lambda x : x\*2 , number))  
numberTwo = [x for x in range(101,201)]   
addnumber = [x+y for x in doubleNumber for y in numberTwo] #这个列表推导式不能代替上面的lambda  
print(addnumber)  
print (dict([(x,y) for x in doubleNumber for y in numberTwo])) #使用这个字典显示了addnumber = [x+y for x in doubleNumber for y in numberTwo]的计算过程

### 3.5.4 lamdaExpression(lambda表达式reduce)

#1.4 lamda 表达式对列表的处理，reduce  
from functools import reduce  
number = [x for x in range(1,101)] #列表的推导式  
sumNumber = reduce(lambda x,y : x+y , number) #使用lambda对原有列表中"每一个"元素进行"累计操作"  
print("列表中所有数字的和是:" + str(sumNumber))

### 3.5.5 lamdaExpression(lambda表达式应用)

# 1.5 if 语句的三元运算  
peoples = ['people01', 'people02', 'people03', 'people04', 'man05', 'people06', 'people07', 'people08','people09', 'people10']  
womens = []  
for people in peoples:  
 if people[0:6] == 'people':  
 womens.append('women' + people[6:])  
 else:  
 womens.append(people)  
print(womens)  
  
  
peoples = ['people01', 'people02', 'people03', 'people04', 'man05', 'people06', 'people07', 'people08','people09', 'people10']  
womens = []  
for people in peoples:  
 womens.append('women' + people[6:]) if people[0:6] == 'people' else womens.append(people) #if的三元运算  
print(womens)  
  
#lamda 表达式对字符串列表的处理  
peoples = ['people01', 'people02', 'people03', 'people04', 'man05', 'people06', 'people07', 'people08','people09', 'people10']  
tempWomen = list(filter(lambda people:people[0:6] == 'people',peoples))  
tempMan = list(filter(lambda people:people[0:6] != 'people',peoples))  
womens = list(map(lambda people:'women' + people[6:],tempWomen))+tempMan #字符串列表的追加  
print(womens)

## 3.6 regularExpresssion

### 3.6.1 regularExpresssion(正则表达式初步)

# 1.1 正则表达式的模块导入和基本用法  
text = input('请输入一串文本：') #一个判定用户输入有效手机号的例子  
if len(text) == 11: #位数必须正确  
 try:  
 if int(text) > 10000000000 and int(text) < 10000000000: #整数且去除0起头的输入  
 print('您输入了一个有效的电话号码：' + text)  
 else:  
 print('您输入了一个无效的电话号码：' + text)  
 except:  
 print('您输入了一个无效的电话号码：' + text)  
else:  
 print('您输入了一个无效的电话号码：' + text)  
  
  
import re #导入正则表达式处理模块re  
text = input('请输入一串文本：') #一个判定用户输入有效手机号的例子  
phoneNumberCheckRE = re.compile(r'\d\d\d\d\d\d\d\d\d\d\d') #匹配任意数字，等价于 [0-9]  
matchObject = phoneNumberCheckRE.search(text)  
if matchObject == None: #没有匹配则对象为none  
 print('您输入了一个无效的电话号码：' + text)  
else:  
 print('您输入了一个有效的电话号码：' + text)

### 3.6.2 regularExpresssion(正则表达式应用)

# 1.2 更加精确的电话匹配  
import re #导入正则表达式处理模块re  
text = input('请输入一串文本：') #一个判定用户输入有效手机号的例子  
phoneNumberCheckRE = re.compile(r'[1][35]\d{9}') #第一位1可以匹配，第二位3或者5可以匹配，也可以使用'[1][35]\d{9}'  
matchObject = re.match(phoneNumberCheckRE,text)  
# phoneNumberCheckRE.search(text)  
if matchObject: #没有匹配则对象为none  
 print('您输入了一个有效的电话号码：' + matchObject.group(0)) #不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。  
else:  
 print('您输入了一个无效的电话号码：' + text)

### 3.6.3 regularExpresssion(正则表达式应用改进)

# # 1.3 座机号的匹配  
import re #导入正则表达式处理模块re  
# text = input('请输入一串文本：') #一个判定用户输入有效手机号的例子  
# # \(? ?表示括号可有可无  
# # (0\d{2,3}) 0\*\*或0\*\*\*,括号代表第一个匹配分组  
# # [), ,-] 表示')'、'-'、' '都是作为分隔符的匹配  
# # (\d{7,8}) 7位或8位的号码  
# phoneNumberCheckRE = re.compile(r"\(?(0\d{2,3})[), ,-](\d{7,8})") #使用括号进行分组，第一个括号内的为group(1)，以此类推  
# matchObject = re.match(phoneNumberCheckRE,text)  
# # phoneNumberCheckRE.search(text)  
# if matchObject: #没有匹配则对象为none  
# print('您输入了一个有效的座机号码：' + matchObject.group(0))#不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。  
# print('您输入的区号是：'+matchObject.group(1))  
# print('您输入的电话号码是：'+matchObject.group(2))  
# else:  
# print('您输入了一个无效的电话号码：' + text)  
  
# cellValueRE = re.compile(r"(\d{1,2,3})(.\*)")  
  
cellValueRE = re.compile(r"(^\d+)") # 字符串以数字开始作为匹配项  
  
cellValue = '12334 中国12社会科3434343学999'  
matchObject = re.match(cellValueRE,cellValue)  
if matchObject:  
 if matchObject.group(1):  
 print(matchObject.group(1))  
  
cellValueRE = re.compile(r"\d+$") # 字符串以数字结束作为匹配项  
numbers = cellValueRE.findall(cellValue) #使用每一个匹配项作为列表的元素  
print(numbers)

### 3.6.4 regularExpresssion(正则表达式分词清洗)

# 1.4 分词清洗的精简版本  
import re #导入正则表达式处理模块re  
text = input('请输入一串文本：') #输入要清洗的文本  
#[',\",\,,:, ,]\* 去除" ' : 及空格，\*号表示匹配0个或者多个  
wordRE = re.compile(r"[',\",\,,:, ,]\*([a-z]{1,20})")  
matchObject = re.match(wordRE,text)  
if matchObject:  
 print('清理前的分词为：' + text)  
 print('清理后的分词为：' + matchObject.group(1))  
else:  
 print('无法进行分词匹配！！！')

## 3.7 dataVisualization

### 3.7.1 dataVisualization(pyecharts数据可视化)

#字符串的处理  
sentence = "What are you doing? I am working."  
print(sentence)  
sentence = sentence.replace('?',' ') #使用replace将？替换成空格  
print(sentence)  
sentence = sentence.replace('.',' ') #使用replace将.替换成空格  
print(sentence)  
words = sentence.split() #split方法以空格作为分隔符将字符串拆分成单词列表  
print(words)  
  
#对英文文本文件的分词处理  
import os  
fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderDir + "\\wordsAnalysis.txt" #使用字符串拼接，得到需要打开文件的绝对路径  
with open(fileFullPath,encoding = 'utf-8') as fileObject: #使用绝对路径打开文件，并将文件作为一个对象fileObject  
 fileContent = fileObject.read() #读取文件对象的内容，并赋值给变量fileContent  
  
  
# print(type(fileContent))  
#将标点符号替换为空格  
fileContent = fileContent.replace('?',' ')  
fileContent = fileContent.replace(';',' ')  
fileContent = fileContent.replace('.',' ')  
fileContent = fileContent.replace(',',' ')  
fileContent = fileContent.replace(':',' ')  
fileContent = fileContent.replace('"',' ')  
fileContent = fileContent.replace("'",' ')  
fileContent = fileContent.replace('(',' ')  
fileContent = fileContent.replace(')',' ')  
fileContent = fileContent.replace("’",' ')  
fileContent = fileContent.replace("“",' ')  
fileContent = fileContent.replace("”",' ')  
words = fileContent.lower().split() #split方法使用空格作为分隔符将字符串fileContent拆分成列表  
# print(words)  
# print(type(words))  
# print(len(words))  
  
for word in words:  
 if len(word) == 1 and word != "i": #去除单词长度为1，且不是大写I的单词  
 words.remove(word)  
# print(words)  
# print(type(words))  
# print(len(words))  
  
wordsUnique = [] #初始化空列表wordsUnique，用于存储words列表中出现的不重复单词  
for word in words:  
 if wordsUnique.count(word) == 0: #如果在wordsUnique列表中找不到单词word就加入该列表  
 wordsUnique.append(word)  
# print(wordsUnique)  
# print(len(wordsUnique))  
  
wordsCount = {}  
for word in wordsUnique:  
 wordsCount[word] = words.count(word) #将words列表中出现word的次数作为值，word作为键加入字典wordsCount  
print(len(wordsCount))  
wordsCount = {wordUnique:words.count(wordUnique) for wordUnique in wordsUnique} #使用生成器模式生成词频字典  
print(len(wordsCount))  
  
  
wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #对字典的value进行排序  
wordsCountDict = dict(wordsCountOrder) #将元祖转换为字典  
# print(wordsCountDict)  
  
from pyecharts import Bar #引入数据可视化模块pyechart中的Bar直方图  
bar\_x = []  
bar\_y = []  
for x,y in wordsCountDict.items():#遍历已排序的字典wordsCountDict的键、值  
 if y > 4: #将词频出现5次及以上的单词进行显示  
 bar\_x.append(x)  
 bar\_y.append(y)  
bar = Bar('words count','English words count')  
bar.use\_theme('dark')  
bar.add('文章1词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.add('文章2词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.add('文章3词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.render('wordsCount.html')  
  
from pyecharts import WordCloud  
# words = list(wordsCountDict.keys())  
# counts = list(wordsCountDict.values())  
wordCloud = WordCloud(width = 900,height = 600)  
wordCloud.add('英文单词词频',bar\_x,bar\_y,word\_size\_range = [60,180])  
wordCloud.render('wordsCloud.html')

### 3.7.2 dataVisualization(matplotlib线性图)

# 1.1 使用matplotlib绘制可视化图形,线性  
import matplotlib.pyplot as plt  
import wordsAnalysisFunction as wordsProcess  
  
fileFullPath = './PythonStudy/020进阶部分/060dataVisualization/wordsAnalysis.txt'  
  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,5)  
  
input\_X = []  
input\_Y = []  
input\_labels = []  
  
number = 1  
  
for word,count in sorted(wordsCount.items(),key = lambda item:item[1]):  
 if count > 1:  
 input\_Y.append(count)  
 input\_X.append(number)  
 input\_labels.append(word)  
 number += 1  
  
plt.plot(input\_X, input\_Y, linewidth=1) #plt.plot(x,y,format\_string,\*\*kwargs)  
plt.show()

### 3.7.3 dataVisualization(matplotlib点状图)

# 1.3 使用matplotlib绘制可视化图形,散点  
import matplotlib.pyplot as plt  
import wordsAnalysisFunction as wordsProcess  
  
fileFullPath = './PythonStudy/020进阶部分/060dataVisualization/wordsAnalysis.txt'  
  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,5)  
  
input\_X = []  
input\_Y = []  
input\_labels = []  
  
number = 1  
  
for word,count in sorted(wordsCount.items(),key = lambda item:item[1]):  
 if count > 1:  
 input\_Y.append(count)  
 input\_X.append(number)  
 input\_labels.append(word)  
 number += 1  
  
plt.scatter(input\_X, input\_Y) #plt.plot(x,y,format\_string,\*\*kwargs)  
plt.show()

### 3.7.4 dataVisualization(pyecharts柱状图)

#2.1 pyecharts的数据可视化  
from pyecharts import Bar  
bar = Bar("我的第一个图表", "这里是副标题")  
bar.add("服装", ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"], [5, 20, 36, 10, 75, 90])  
# bar.print\_echarts\_options() # 该行只为了打印配置项，方便调试时使用  
bar.render('./PythonStudy/020进阶部分/060dataVisualization/210pyechartsBarBasic.html') # 生成本地 HTML 文件

### 3.7.5 dataVisualization(pyecharts柱状图词频表达)

#2.2 pyecharts对分词结果的处理  
#字符串的处理  
sentence = "What are you doing? I am working."  
print(sentence)  
sentence = sentence.replace('?',' ') #使用replace将？替换成空格  
print(sentence)  
sentence = sentence.replace('.',' ') #使用replace将.替换成空格  
print(sentence)  
words = sentence.split() #split方法以空格作为分隔符将字符串拆分成单词列表  
print(words)  
  
#对英文文本文件的分词处理  
import os  
fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderDir + "\\wordsAnalysis.txt" #使用字符串拼接，得到需要打开文件的绝对路径  
with open(fileFullPath,encoding = 'utf-8') as fileObject: #使用绝对路径打开文件，并将文件作为一个对象fileObject  
 fileContent = fileObject.read() #读取文件对象的内容，并赋值给变量fileContent  
  
  
# print(type(fileContent))  
#将标点符号替换为空格  
fileContent = fileContent.replace('?',' ')  
fileContent = fileContent.replace(';',' ')  
fileContent = fileContent.replace('.',' ')  
fileContent = fileContent.replace(',',' ')  
fileContent = fileContent.replace(':',' ')  
fileContent = fileContent.replace('"',' ')  
fileContent = fileContent.replace("'",' ')  
fileContent = fileContent.replace('(',' ')  
fileContent = fileContent.replace(')',' ')  
fileContent = fileContent.replace("’",' ')  
fileContent = fileContent.replace("“",' ')  
fileContent = fileContent.replace("”",' ')  
words = fileContent.lower().split() #split方法使用空格作为分隔符将字符串fileContent拆分成列表  
# print(words)  
# print(type(words))  
# print(len(words))  
  
for word in words:  
 if len(word) == 1 and word != "i": #去除单词长度为1，且不是大写I的单词  
 words.remove(word)  
# print(words)  
# print(type(words))  
# print(len(words))  
  
wordsUnique = [] #初始化空列表wordsUnique，用于存储words列表中出现的不重复单词  
for word in words:  
 if wordsUnique.count(word) == 0: #如果在wordsUnique列表中找不到单词word就加入该列表  
 wordsUnique.append(word)  
# print(wordsUnique)  
# print(len(wordsUnique))  
  
wordsCount = {}  
for word in wordsUnique:  
 wordsCount[word] = words.count(word) #将words列表中出现word的次数作为值，word作为键加入字典wordsCount  
# print(wordsCount)  
  
wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #对字典的value进行排序  
wordsCountDict = dict(wordsCountOrder) #将元祖转换为字典  
# print(wordsCountDict)  
  
from pyecharts import Bar #引入数据可视化模块pyechart中的Bar直方图  
bar\_x = []  
bar\_y = []  
for x,y in wordsCountDict.items():#遍历已排序的字典wordsCountDict的键、值  
 if y > 4: #将词频出现5次及以上的单词进行显示  
 bar\_x.append(x)  
 bar\_y.append(y)  
bar = Bar('words count','English words count')  
bar.use\_theme('dark')  
bar.add('文章1词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.add('文章2词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.add('文章3词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.render('wordsCount.html')  
  
from pyecharts import WordCloud  
# words = list(wordsCountDict.keys())  
# counts = list(wordsCountDict.values())  
wordCloud = WordCloud(width = 900,height = 600)  
wordCloud.add('英文单词词频',bar\_x,bar\_y,word\_size\_range = [60,180])  
wordCloud.render('wordsCloud.html')

### 3.7.6 dataVisualization(pyecharts柱状图主题词频表达)

#2.3 pyecharts使用主题(pip install echarts-themes-pypkg)  
from pyecharts import Bar  
import wordsAnalysisFunction as wordsProcess  
import os  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/wordsAnalysis.txt'  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,5)  
input\_X = []  
input\_Y = []  
for word, count in wordsCount.items():  
 if count > 5:  
 input\_Y.append(count)  
 input\_X.append(word)  
  
bar = Bar('分词统计的范例','英文分词')  
bar.use\_theme("dark") #使用黑色主题。dark,chalk....等等  
bar.add('文章词频',input\_X,input\_Y)  
# bar.add('文章词频',input\_X,input\_Y,is\_more\_utils=True) #更加强大的工具选项，支持局部缩放和折线图  
bar.render( './PythonStudy/020进阶部分/060dataVisualization/230pyechartsTheme.html') # 生成本地 HTML 文件

### 3.7.7 dataVisualization(pyecharts柱状图词频表达统计属性)

#2.4 pyecharts使用字典转换成有序列表  
from pyecharts import Bar  
import wordsAnalysisFunction as wordsProcess  
import os  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/wordsAnalysis.txt'  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,5)  
  
bar = Bar('分词统计的范例','英文分词')  
bar.use\_theme("dark") #使用黑色主题。dark,chalk....等等  
bar.add('文章词频',list(wordsCount.keys()),list(wordsCount.values()),mark\_line = ['average'],mark\_point = ['max','min']) #使用字典转换为列表  
bar.render( './PythonStudy/020进阶部分/060dataVisualization/240pyechartsBarMore.html') # 生成本地 HTML 文件

### 3.7.8 dataVisualization(pyecharts词云词频表达)

#1.5 pyecharts使用词云显示词频  
from pyecharts import WordCloud  
import wordsAnalysisFunction as wordsProcess  
import os  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/wordsAnalysis.txt'  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsCount = wordsProcess.analyzeWordsCount(wordsCleaned,5)  
  
words = list(wordsCount.keys())  
counts = list(wordsCount.values())  
wordCloud = WordCloud(width = 1300, height = 600)  
wordCloud.add('词云可视化',words,counts,word\_size\_range = [20,300]) #word\_size\_range 最少的频次和最大的频次的图像比例区间  
wordCloud.show\_config()  
wordCloud.render('./PythonStudy/020进阶部分/060dataVisualization/250pyechartsWordCloud.html')

### 3.7.9 dataVisualization(pyecharts词云中文词频表达)

# 1.1 中文分词模块jieba  
import os  
import jieba  
words = '我们今天早点去学校上课，因为要打扫卫生，今天上课内容是python程序设计。'  
words\_list = jieba.cut(words, cut\_all=True)  
print("-".join(words\_list))  
words\_list = jieba.cut(words, cut\_all=False)  
print(" ".join(words\_list))  
words\_list = jieba.cut\_for\_search(words)  
print("/".join(words\_list))  
  
# 1.2 对文本的分词  
import jieba  
from pyecharts import WordCloud  
import os  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/wordsAnalysis.txt'  
with open(os.path.join('./PythonStudy/020进阶部分/060dataVisualization/','wordCHAnalysis.txt'),'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 jieba.add\_word('大数据') #添加用户自己定义的字典  
 jieba.add\_word('云计算')  
 jieba.add\_word('区块链')  
 words\_cut = jieba.cut(contents, cut\_all=True)  
  
words\_list = list(words\_cut)  
print(list(words\_list))  
  
wordsUnique = []  
wordsCount = {}  
minCount = 4  
for word in words\_list:  
 if word not in wordsUnique:  
 wordsUnique.append(word)  
for wordUnique in wordsUnique:  
 wordCount = words\_list.count(wordUnique)  
 if wordCount >= minCount: #此处利用参数来选择最小的词频  
 wordsCount[wordUnique] = wordCount  
wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #使用该排序将字典变成有序的元组  
wordsCountDict = dict(wordsCountOrder) #将有序的元组转换为字典  
del wordsCountDict['的'] #去除一些无意义的统计词频  
del wordsCountDict['与']  
del wordsCountDict['和']  
del wordsCountDict['了']  
del wordsCountDict['年']  
del wordsCountDict['个']  
del wordsCountDict['从']  
del wordsCountDict['将']  
del wordsCountDict['在']  
del wordsCountDict['等']  
del wordsCountDict['是']  
del wordsCountDict['']  
print(wordsCountDict)  
  
  
words = list(wordsCountDict.keys())  
counts = list(wordsCountDict.values())  
wordCloud = WordCloud(width = 1300, height = 600)  
wordCloud.add('词云可视化',words,counts,word\_size\_range = [20,100]) #word\_size\_range 最少的频次和最大的频次的图像比例区间  
wordCloud.render(os.path.join('./PythonStudy/020进阶部分/060dataVisualization/','260pyechartsWordCloud.html'))

### 3.7.10 dataVisualization(使用语言高级特性的分词方式)

import os  
import wordsAnalysisFunction as wordsProcess  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileFullPath = fileFolderPath + '/wordsAnalysis.txt'  
print(fileFolderPath)  
  
wordsCleaned = wordsProcess.textDivide(fileFullPath)  
wordsUniqueSet = set(wordsCleaned) #利用set中元素唯一性的特点，保留了列表中唯一的单词  
wordsUniqueList = list(wordsUniqueSet)  
# words = nltk.text.Text(cotent)  
print(len(wordsCleaned))  
print(len(wordsUniqueSet))  
print(len(wordsUniqueList))  
  
wordsFrequencyDic = {word : wordsCleaned.count(word) for word in wordsUniqueList} #此处使用了字典推导式  
print(wordsFrequencyDic)  
print(len(wordsFrequencyDic))

### 3.7.11 dataVisualization(中英文分词的迭代处理)

#绝对路径  
#转义符'\\'和读取中文的编码'UTF-8',linux系统使用斜杠'/'  
with open('D:\\PythonStudy-V1.0\\PythonStudy\\020进阶部分\\060dataVisualization\\fileRead.txt',encoding='UTF-8') as fileObject:   
 contents = fileObject.read()  
 print(contents)  
   
# #相对路径  
# #转义符'\\'和读取中文的编码'UTF-8',linux系统使用斜杠'/'  
# with open('./fileRead.txt',encoding='UTF-8') as fileObject:   
# contents = fileObject.read()  
# print(contents)  
  
# 使用os模块获取当前文件和文件夹的绝对路径  
import os  
filePath = os.path.abspath(\_\_file\_\_) #当前文件的绝对路径  
print(filePath)  
  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
print(fileFolderPath)   
  
#使用os模块使用绝对路径读取文件  
import os  
fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
fileName = "fileRead.txt"  
fileFullPath = fileFolderPath + "\\" + fileName  
with open(fileFullPath,encoding='UTF-8') as fileObject:   
 contents = fileObject.read()  
 print(contents)  
  
#使用封装的函数进行文件读取  
import fileReadFunc  
with open(fileReadFunc.fileOpen("fileRead.txt"),encoding='UTF-8') as fileObject:   
 contents = fileObject.read()  
 print(contents)  
  
  
# 1.3 文件的逐行读取和内容预处理  
import fileReadFunc  
lineNumber = 1  
lineLength = 0   
with open(fileReadFunc.fileOpen("fileRead.txt"),encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1  
  
# 1.5 基本的文件内容查找  
import fileReadFunc  
lineNumber = 1  
lineLength = 0   
lines = []  
with open(fileReadFunc.fileOpen("fileRead.txt"),encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 lines.append(lineProcessed)  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1  
charFind = input('请输入你要查找的字符：')  
lineNumber = 0  
for line in lines:  
 charCount = line.count(charFind)  
 lineNumber +=1   
 print('第' + str(lineNumber) + '段找到' + "'" + charFind + "'" + '共计' + str(charCount) + '个')  
  
  
  
# 1.5 基本的文件内容查找，结合while的改进版本  
import fileReadFunc  
lineNumber = 1  
lineLength = 0   
lines = []  
with open(fileReadFunc.fileOpen("wordsAnalysis.txt"),encoding='UTF-8') as fileObject:  
 for line in fileObject:  
 lineProcessed = line.strip() #删除首尾的空格，预处理  
 if (len(lineProcessed) != 0): #将空行处理掉  
 lines.append(lineProcessed)  
 print('第'+str(lineNumber)+'行，' + '长度为：'+str(len(lineProcessed)))  
 print(lineProcessed)  
 lineNumber += 1  
while True:  
 charFind = input('请输入你要查找的字符(输入"quit"结束分析)：')  
 if charFind != 'quit':  
 lineNumber = 0  
 charCountSum = 0  
 for line in lines:  
 charCount = line.count(charFind) #计算每一行找到的字符个数  
 charCountSum = charCountSum + charCount #计算找到字符的合计数  
 lineNumber = lineNumber + 1   
 print('第' + str(lineNumber) + '行找到' + "'" + charFind + "'" + '共计' + str(charCount) + '个')  
 print('找到' + "'" + charFind + "'" + '共计' + str(charCountSum) + '个')   
 else:  
 break  
  
#英文分词的基础  
contents = "I’m coming back tomorrow to see how you are getting along,You are the good man."  
words = contents.split()  
print(words)  
contents = contents.replace("'",' ')  
contents = contents.replace(",",' ')  
contents = contents.replace(".",' ')  
words = contents.split()  
print(words)  
  
contents = "I’m coming back tomorrow to see how you are getting along,You are the good man."  
words = contents.replace("'",' ').replace(",",' ').replace(".",' ').split()  
print(words)  
   
import fileReadFunc  
with open(fileReadFunc.fileOpen("wordsAnalysis.txt"),encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 contents = contents.replace("'",' ')  
 contents = contents.replace(",",' ')  
 contents = contents.replace(".",' ')  
 contents = contents.replace("?",' ')  
 contents = contents.replace("!",' ')  
 words = contents.split()  
print(words)  
  
wordsCleaned = []  
for word in words:  
 if (len(word) != 1):  
 wordsCleaned.append(word.lower())  
print(wordsCleaned)  
  
print(len(words))  
print(len(wordsCleaned))  
  
wordsUniqueSet = set(wordsCleaned)  
wordsUnique = list(wordsUniqueSet)  
print(len(wordsUnique))  
  
  
wordsCount = {}  
for word in wordsUnique:  
 if wordsCleaned.count(word) > 1:  
 wordsCount[word] = wordsCleaned.count(word)  
print(wordsCount)  
  
wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #使用该排序将字典变成有序的元组  
wordsCountDict = dict(wordsCountOrder)   
print(wordsCountDict)  
  
#写入CSV文件  
import csv  
with open(fileReadFunc.fileOpen("fileAnalysisUpper.csv"),'w',newline = '',encoding='UTF-8') as csvFile:  
 csvWrite = csv.writer(csvFile)  
 for row in wordsCountDict.items():  
 csvWrite.writerow(row)  
  
  
from pyecharts import Bar #引入数据可视化模块pyechart中的Bar直方图  
bar\_x = []  
bar\_y = []  
for x,y in wordsCountDict.items():#遍历已排序的字典wordsCountDict的键、值  
 if y > 4: #将词频出现5次及以上的单词进行显示  
 bar\_x.append(x)  
 bar\_y.append(y)  
bar = Bar('words count','English words count')  
bar.use\_theme('dark')  
bar.add('文章词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
bar.render('wordsCount.html')  
  
from pyecharts import WordCloud  
bar\_x = list(wordsCountDict.keys())  
bar\_y = list(wordsCountDict.values())  
wordCloud = WordCloud(width = 900,height = 600)  
wordCloud.add('英文单词词频',bar\_x,bar\_y,word\_size\_range = [60,180])  
wordCloud.render('wordsCloud.html')  
  
#使用函数封装的分词可视化  
import wordsProcess  
wordsCleaned = []  
with open(wordsProcess.fileOpen("wordsAnalysis.txt"),encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 wordsCleaned = wordsProcess.wordsClean(contents)  
wordsCountDict = wordsProcess.wordsStatic(wordsCleaned,9)  
wordsProcess.resultShow('bar',wordsCountDict,'英文词频','英文词频统计范例','word and count')  
wordsProcess.resultShow('wordcloud',wordsCountDict,'英文词频')  
  
  
#中文分词的基本处理  
import jieba  
words = '我们今天早点去学校上课，因为要打扫卫生，今天上课内容是python程序设计。'  
words\_list = jieba.cut(words, cut\_all=False)  
print(list(words\_list))  
words\_list = jieba.cut(words, cut\_all=True)  
print(list(words\_list))  
  
#中文分词对文件的处理  
import jieba  
import wordsProcess  
with open(wordsProcess.fileOpen("wordCHAnalysis.txt"),encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
jieba.add\_word('大数据') #添加用户自己定义的字典  
jieba.add\_word('云计算')  
jieba.add\_word('区块链')  
jieba.add\_word('新技术')  
jieba.add\_word('C端')  
jieba.add\_word('B端')  
wordslist = list(jieba.cut(contents, cut\_all=False))  
  
wordsSet = set(wordslist)  
wordsUniquelist = list(wordsSet)  
print(wordsUniquelist)  
print(len(wordslist))  
print(len(wordsUniquelist))  
  
for word in wordsUniquelist:  
 if len(word.strip()) == 1:  
 wordsUniquelist.remove(word)  
print(len(wordsUniquelist))  
print(wordsUniquelist)  
  
wordsCountDict = {}  
for word in wordsUniquelist:  
 if wordslist.count(word) > 2:  
 wordsCountDict[word] = wordslist.count(word)  
  
wordsCountOrderDict = dict(sorted(wordsCountDict.items(),key = lambda item:item[1]))  
print(wordsCountOrderDict)  
  
wordsProcess.resultShow('bar',wordsCountOrderDict,'中文词频','中文词频统计范例','单词和词频')  
wordsProcess.resultShow('wordcloud',wordsCountOrderDict,'中文词频')  
  
  
#函数封装的中文词频处理程序  
import wordsProcess  
wordsCuted = []  
with open(wordsProcess.fileOpen("wordCHAnalysis.txt"),encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 wordsCuted = wordsProcess.chWordsCut(contents)  
wordsUniquelist = wordsProcess.chWordsUniqueClean(wordsCuted)  
wordsCountOrderDict = wordsProcess.chWordsStatic(wordsUniquelist,wordsCuted,3)  
  
wordsProcess.resultShow('bar',wordsCountOrderDict,'中文词频','中文词频统计范例','单词和词频')  
wordsProcess.resultShow('wordcloud',wordsCountOrderDict,'中文词频')

### 3.7.12 funcUse

#使用函数封装对文件的读取、分词、清理及统计词频的功能  
import funcWordsProcess  
fileName = "\\wordsAnalysis.txt"  
fileContent = funcWordsProcess.fileRead(fileName)  
wordsCleaned = funcWordsProcess.fileClean(fileContent)  
wordsUnique = funcWordsProcess.fileWordsUnique(wordsCleaned)  
wordsCountDict = funcWordsProcess.fileWordsCount(wordsCleaned,wordsUnique)  
  
help(funcWordsProcess.fileRead) #该代码查看函数的文档内容  
  
#使用模块中函数封装后的函数  
wordsCountDict = funcWordsProcess.fileWordsCountForEN(fileName)  
print(wordsCountDict)  
print(len(wordsCountDict))  
  
#使用封装的函数完成直方图和词云的可视化表达  
import funcWordsProcess  
import funcWordsCountVisualization  
fileName = "\\wordsAnalysis.txt"  
wordsCountDict = funcWordsProcess.fileWordsCountForEN(fileName)  
funcWordsCountVisualization.drawWordsCountBar(wordsCountDict)  
funcWordsCountVisualization.drawWordsCountCloud(wordsCountDict)

### 3.7.13 funcWordsCountVisualization

from pyecharts import Bar  
from pyecharts import WordCloud  
import os  
  
def getFilePathFolder():  
 fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
 filePathFolder= fileFolderDir+"\\"  
 return(filePathFolder)  
  
def getValueXY(wordsCountDict):  
 bar\_x = []  
 bar\_y = []  
 for x,y in wordsCountDict.items():#遍历已排序的字典wordsCountDict的键、值  
 if y > 4: #将词频出现5次及以上的单词进行显示  
 bar\_x.append(x)  
 bar\_y.append(y)  
 return([bar\_x,bar\_y])  
  
def drawWordsCountBar(wordsCountDict):  
 bar\_x = getValueXY(wordsCountDict)[0]  
 bar\_y = getValueXY(wordsCountDict)[1]  
  
 bar = Bar('词频统计','英文词频统计')  
 bar.use\_theme('dark')  
 bar.add('文章词频',bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
 filePathFolder = getFilePathFolder()  
 bar.render(filePathFolder + 'wordsCount.html')  
  
def drawWordsCountCloud(wordsCountDict):  
 bar\_x = getValueXY(wordsCountDict)[0]  
 bar\_y = getValueXY(wordsCountDict)[1]  
  
 wordCloud = WordCloud(width = 900,height = 600)  
 wordCloud.add('英文单词词频',bar\_x,bar\_y,word\_size\_range = [60,180])  
 filePathFolder = getFilePathFolder()  
 wordCloud.render(filePathFolder + 'wordsCloud.html')

### 3.7.14 funcWordsProcess

def fileRead(fileName):   
 '''  
 本函数将打开与本文件同一文件夹路径下的文件进行读取  
 传入参数为文件的全名[string]  
 传出参数为读取的文件内容[string]  
 '''  
 import os  
 fileFolderDir = os.path.dirname(\_\_file\_\_) #使用os模块的函数拿到正在当前文件所在文件夹的绝对路径  
 fileFullPath = fileFolderDir + fileName #使用字符串拼接，得到需要打开文件的绝对路径  
 with open(fileFullPath,encoding = 'utf-8') as fileObject: #使用绝对路径打开文件，并将文件作为一个对象fileObject  
 fileContent = fileObject.read() #读取文件对象的内容，并赋值给变量fileContent  
 return fileContent  
  
def fileClean(fileContent):  
 '''  
 本函数将读取的文件内容形成清洗后的单词列表  
 传入参数为读取的文件内容[string]  
 传出的内容为单词列表[list]  
 '''  
 #将标点符号替换为空格  
 fileContent = fileContent.replace('?',' ')  
 fileContent = fileContent.replace(';',' ')  
 fileContent = fileContent.replace('.',' ')  
 fileContent = fileContent.replace(',',' ')  
 fileContent = fileContent.replace(':',' ')  
 fileContent = fileContent.replace('"',' ')  
 fileContent = fileContent.replace("'",' ')  
 fileContent = fileContent.replace('(',' ')  
 fileContent = fileContent.replace(')',' ')  
 fileContent = fileContent.replace("’",' ')  
 fileContent = fileContent.replace("“",' ')  
 fileContent = fileContent.replace("”",' ')  
 wordsCleaned = fileContent.lower().split() #split方法使用空格作为分隔符将字符串fileContent拆分成列表  
 for word in wordsCleaned:  
 if len(word) == 1 and word != "i": #去除单词长度为1，且不是大写I的单词  
 wordsCleaned.remove(word)  
 return wordsCleaned  
  
def fileWordsUnique(wordsCleaned):  
 '''  
 本函数将从单词列表获得单词列表中出现的单词  
 传入参数为读取的单词列表[list]  
 传出的内容为单词列表中出现的不重复单词列表[list]  
 '''  
 #第一代获取不重复单词列表的处理代码  
 wordsUnique = [] #初始化空列表wordsUnique，用于存储words列表中出现的不重复单词  
 for word in wordsCleaned:  
 if wordsUnique.count(word) == 0: #如果在wordsUnique列表中找不到单词word就加入该列表  
 wordsUnique.append(word)  
  
 # #第二代获取不重复单词列表的处理代码  
 # wordsUniqueSet = set(wordsCleaned)  
 # wordsUnique = list(wordsUniqueSet)  
 return wordsUnique  
  
def fileWordsCount(wordsCleaned,wordsUnique):  
 '''  
 本函数将单词列表中的单词进行计数  
 传入参数为读取的单词列表wordsCleaned[list]，不重复单词列表wordsUnique[list]  
 传出的内容为单词及计数的字典[dict]  
 '''  
 wordsCount = {}  
 for word in wordsUnique:  
 wordsCount[word] = wordsCleaned.count(word) #将words列表中出现word的次数作为值，word作为键加入字典wordsCount  
 wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #对字典的value进行排序  
 wordsCountDict = dict(wordsCountOrder) #将元祖转换为字典  
 return wordsCountDict  
  
def fileWordsCountForEN(fileName):  
 '''  
 该函数对四个函数进行了封装  
 1.fileRead  
 2.fileClean  
 3.fileWordsUnique  
 4.fileWordsCount  
 传出的内容为单词及计数的字典[dict]  
 '''  
 fileContent = fileRead(fileName)  
 wordsClean = fileClean(fileContent)  
 wordsUnique = fileWordsUnique(wordsClean)  
 wordsCountDict = fileWordsCount(wordsClean,wordsUnique)  
 return wordsCountDict

### 3.7.15 wordsAnalysisFunction

import re #导入正则表达式处理模块re  
def wordMatch(wordForClean): #定义单个分词的清洗  
 # wordRE = re.compile(r"[‘,“,(]\*([a-z]{1,20})")  
 wordRE = re.compile(r"[^a-z]\*([a-z]{1,20})")  
 matchObject = re.match(wordRE,wordForClean)  
 return matchObject  
  
def wordsClean(textForDivide): #定义单词列表的清洗  
 wordsForClean = textForDivide.split()  
 wordsCleaned = []  
 for wordForClean in wordsForClean:  
 matchObject = wordMatch(wordForClean.lower()) #全部转成小写的处理  
 if matchObject:  
 wordsCleaned.append(matchObject.group(1))  
 return wordsCleaned  
  
def textDivide(fileFullPath): #定义拆分单词的函数  
 wordsCleaned = []  
 with open(fileFullPath,'r',encoding='UTF-8') as fileObject:  
 contents = fileObject.read()  
 wordsCleaned = wordsClean(contents)  
 return wordsCleaned  
   
def analyzeWordsCount (wordsCleaned,minCount): #定义基本的统计函数  
 wordsUnique = []  
 wordsCount = {}  
 for word in wordsCleaned:  
 if word not in wordsUnique:  
 wordsUnique.append(word)  
 for wordUnique in wordsUnique:  
 wordCount = wordsCleaned.count(wordUnique)  
 if wordCount >= minCount: #此处利用参数来选择最小的词频  
 wordsCount[wordUnique] = wordCount  
 wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #使用该排序将字典变成有序的元组  
 wordsCountDict = dict(wordsCountOrder) #将有序的元组转换为字典  
 return wordsCountDict #返回字典

### 3.7.16 wordsProcess

import os  
from pyecharts import Bar  
from pyecharts import WordCloud  
import jieba  
  
def fileOpen(fileName):  
 fileFolderPath = os.path.dirname(os.path.abspath(\_\_file\_\_)) #当前文件所在文件夹的绝对路径  
 fileFullPath = fileFolderPath + "\\" + fileName  
 return fileFullPath  
  
def wordsClean(contents):  
 contents = contents.replace("'",' ')  
 contents = contents.replace(",",' ')  
 contents = contents.replace(".",' ')  
 contents = contents.replace("?",' ')  
 contents = contents.replace("!",' ')  
 words = contents.split()  
 wordsCleaned = []  
 for word in words:  
 if (len(word) != 1):  
 wordsCleaned.append(word.lower())  
 return wordsCleaned  
  
def wordsStatic(wordsCleaned,number):  
 wordsUniqueSet = set(wordsCleaned)  
 wordsUnique = list(wordsUniqueSet)  
 wordsCount = {}  
 for word in wordsUnique:  
 if wordsCleaned.count(word) > number:  
 wordsCount[word] = wordsCleaned.count(word)  
 wordsCountOrder = sorted(wordsCount.items(),key = lambda item:item[1]) #使用该排序将字典变成有序的元组  
 wordsCountDict = dict(wordsCountOrder)   
 return wordsCountDict  
  
def chWordsCut(contents):  
 jieba.add\_word('大数据') #添加用户自己定义的字典  
 jieba.add\_word('云计算')  
 jieba.add\_word('区块链')  
 jieba.add\_word('新技术')  
 jieba.add\_word('C端')  
 jieba.add\_word('B端')  
 return list(jieba.cut(contents, cut\_all=False))  
  
def chWordsUniqueClean(wordsCuted):  
 wordsSet = set(wordsCuted)  
 wordsUniquelist = list(wordsSet)  
 wordsUniqueCleanlist = []  
 for word in wordsUniquelist:  
 if len(word.strip()) != 1 and len(word.strip()) != 0:  
 wordsUniqueCleanlist.append(word)  
 return wordsUniqueCleanlist  
  
def chWordsStatic(wordsUniquelist,wordslist,number):  
 wordsCountDict = {}  
 for word in wordsUniquelist:  
 if wordslist.count(word) > number:  
 wordsCountDict[word] = wordslist.count(word)  
 wordsCountOrderDict = dict(sorted(wordsCountDict.items(),key = lambda item:item[1]))  
 return wordsCountOrderDict  
  
def resultShow(typeShow,wordsCountDict,name,mainName = '',subName = ''):  
 '''typeShow目前支持bar和wordcloud两种类型，wordsCountDict参数为处理好的词频字典'''  
 bar\_x = list(wordsCountDict.keys())  
 bar\_y = list(wordsCountDict.values())  
 if typeShow == 'bar':  
 bar = Bar(mainName,subName)  
 bar.use\_theme('dark')  
 bar.add(name,bar\_x,bar\_y,mark\_line = ['average'],mark\_point = ['max','min'])  
 bar.render('wordsBar.html')  
 if typeShow == 'wordcloud':  
 wordCloud = WordCloud(width = 1024,height = 768)  
 wordCloud.add(name,bar\_x,bar\_y,word\_size\_range = [20,120])  
 wordCloud.render('wordsCloud.html')

# 4 高级部分

## 4.1 语法最佳实践

### 4.1.1 020enumerate函数使用

#enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。  
#enumerate: 列举，枚举，数  
  
#1.1 获取列表索引的低效做法  
index = 0   
namesList = ['周杰伦','李健','谢霆锋','庾澄庆']  
for name in namesList:  
 print(index,name)  
 index = index + 1  
  
#1.2 使用enumerate获取列表索引的高效做法  
namesList = ['周杰伦','李健','谢霆锋','庾澄庆']  
for index, name in enumerate(namesList):  
 print(index,name)  
  
#1.3 使用enumerate直接获取索引和列表值  
seasons = ['Spring', 'Summer', 'Fall', 'Winter']  
seasonsList= list(enumerate(seasons))  
print(seasonsList)  
  
#1.3 使用enumerate直接获取索引和列表值  
seasons = ['Spring', 'Summer', 'Fall', 'Winter']  
seasonsList = list(enumerate(seasons,1)) #使用第二个参数控制索引的开始  
seasonsDict = dict (enumerate(seasons,1))  
print(seasonsList)  
print(seasonsDict)

### 4.1.2 030zip函数使用

#1.1 zip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。  
namesList = ['周杰伦','李健','谢霆锋','庾澄庆']  
ageList = [40,45,46,53]  
zipped = zip(namesList,ageList)  
for zipedTuple in zipped:  
 print(zipedTuple)  
  
#1.2 对两个大小相等的可迭代对象进行均匀遍历时，使用zip函数是一种常用模式  
namesList = ['周杰伦','李健','谢霆锋','庾澄庆']  
ageList = [40,45,46,53]  
for zipedTuple in zip(namesList,ageList):  
 print(zipedTuple)

### 4.1.3 040序列解包的操作

#1.1 序列解包（sequence unpacking）。这种方法并不限于列表和元组，而是适用于任意序列类型（甚至包括字符串和字节序列）。  
# 只要赋值运算符左边的变量数目与序列中的元素数目相等，你都可以用这种方法将元素序列解包到另一组变量中。  
first,second,third = '小明','小李','小江'  
print(first,second,third)  
  
#1.2 解包还可以利用带星号的表达式获取单个变量中的多个元素，只要它的解释没有歧义即可。  
first,second,third,\*rest = '小明','小李','小江','小赵','小钱','小孙'  
print(first,second,third,rest)  
  
first,\*inner,last = '小明','小李','小江','小赵','小钱','小孙'  
print(first,inner,last)  
  
#1.3 对嵌套序列进行解包。特别是在遍历由序列构成的复杂数据结构时，这种方法非常实用。  
(a, b), (c, d) = (1, 2), (3, 4)  
print(a,b,c,d)

### 4.1.4 050字典的最佳操作

#1.1 使用字典推导具有与列表推导相同的优点。  
# 因此在许多情况下，字典推导要更加高效、更加简短、更加整洁。  
# 对于更复杂的代码而言，需要用到许多if语句或函数调用来创建一个字典，这时最好使用简单的for循环，尤其是它还提高了可读性。  
squares = {number : number\*\*2 for number in range(5)}  
print(squares)  
  
squares = {str(number) : number\*\*2 for number in range(5)}  
print(squares)  
  
#1.2 使用Python标准库的collections模块提供了名为OrderedDict的有序字典。  
from collections import OrderedDict  
squares = OrderedDict((str(number),number) for number in range(5))  
print(squares)  
print(squares.keys())  
  
<<<<<<< HEAD  
#1.3 对字典的排序  
testDict = {'b':3,'a':4,'d':1,'c':2}  
testDictOrdered = sorted(testDict.items(),key = lambda x:x[0],reverse = True)  
print(testDictOrdered)  
testDictOrdered = sorted(testDict.items(),key = lambda x:x[1],reverse = True)  
print(testDictOrdered)  
=======  
#1.3 使用sorted函数对字典进行排序  
fruitDict = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}  
orderByFruit = OrderedDict(sorted(fruitDict.items(), key = lambda item:item[0]))  
orderByNumber = OrderedDict(sorted(fruitDict.items(), key = lambda item:item[1]))  
print(orderByFruit)  
print(orderByNumber)  
  
#1.4 使用sorted函数对字典进行逆排序  
fruitDict = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}  
orderByFruit = OrderedDict(sorted(fruitDict.items(), key = lambda item:item[0]), reversed = True)  
orderByNumber = OrderedDict(sorted(fruitDict.items(), key = lambda item:item[1]), reversed = True)  
print(orderByFruit)  
print(orderByNumber)  
>>>>>>> 863484d0d739c62ed98baf1d5535d474c5faa24c

### 4.1.5 060集合的最佳操作

#1.1 Python的内置集合类型决定了哪一类集合可以作为集合的元素  
#set()：一种可变的、无序的、有限的集合，其元素是唯一的、不可变的（可哈希的）对象。  
#frozenset()：一种不可变的、可哈希的、无序的集合，其元素是唯一的、不可变的（可哈希的）对象。  
#由于frozenset()具有不变性，它可以用作字典的键，也可以作为其他set()和frozenset()的元素。在一个set()或frozenset()中不能包含另一个普通的可变set()  
nameSetOne = set(['小明','小李','小江'])  
nameSetTwo = set(['小赵','小钱','小孙'])  
  
  
#使用可变的set()来充当集合的不可变元素，会引发TypeError  
nameSet = set([nameSetOne,nameSetTwo])  
  
#使用不可变的frozenset()来充当集合的不可变元素  
nameSetOne = frozenset(['小明','小李','小江'])  
nameSetTwo = frozenset(['小赵','小钱','小孙'])  
nameSet = set([nameSetOne,nameSetTwo])  
print(nameSet)  
  
  
#空的集合对象是没有字面值的。空的花括号{}表示的是空的字典字面值。

### 4.1.6 065高级函数的操作(10 map)

#iterable: 可迭代的; 可重复的; 迭代的  
#map的使用：map(function, iterable, ...)  
#map()函数接收两个参数，一个是函数，一个是Iterable，map将传入的函数依次作用到序列的每个元素，并把结果作为新的Iterator返回。  
  
numberList = [number for number in range(1,11)]  
print(numberList)  
  
#1.1 使用for循环来处理列表元素值翻倍  
numberProcessList = []  
for number in numberList:  
 numberProcess = number \* 2  
 numberProcessList.append(numberProcess)  
print(numberProcessList)  
  
  
#1.2 使用map函数来处理列表元素值翻倍  
def multiply(number):  
 return number \* 2  
  
numberProcessList = list(map(multiply, numberList))  
print(numberProcessList)  
  
# 使用map函数来计算球体体积  
import math  
def volumeSphere(radius):  
 return round(4 \* math.pi \* radius\*\*3/3, 2) #python是以双精度(64)位来保存浮点数，多余的位会被截掉,使用round来定义小数位数的精度  
  
volumeSphereList = list(map(volumeSphere, numberList))  
print(volumeSphereList)  
  
#1.3 使用map函数与匿名函数结合来处理列表元素值翻倍  
numberProcessList = list( map( lambda number : number \* 2 , numberList) )  
print(numberProcessList)  
  
# 使用map函数与匿名函数结合来计算球体体积  
volumeSphereList = list(map(lambda radius : round(4 \* math.pi \* radius\*\*3/3, 2), numberList))  
print(volumeSphereList)  
  
#1.4 使用列表推导式来处理列表元素值翻倍  
numberProcessList = [number\*2 for number in numberList]  
print(numberProcessList)  
  
#1.5 对多个列表的操作的传统方式与使用map方式的比较  
numberOneList = [number for number in range(1,11)]  
numberTwolist = list( map( lambda number : round(number \* 0.1, 1), numberOneList ) )   
numberThreeList = list( map( lambda number : number \* 10, numberOneList ) )  
  
#传统方式  
numberSumList = []  
for index in (range(len(numberOneList))): #使用列表长度来进行索引的控制  
 numberSum = numberOneList[index] + numberTwolist[index] + numberThreeList[index]  
 numberSumList.append(numberSum)  
print(numberSumList)  
  
#使用map函数的方式  
numberSumList = list(map( lambda numberOne, numberTwo, numberThree : numberOne + numberTwo + numberThree, numberOneList, numberTwolist, numberThreeList ))  
print(numberSumList)

### 4.1.7 066高级函数的操作(20 filter)

#filter的使用：filter(function, iterable)  
#filter()也接收一个函数和一个序列。和map()不同的是，filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。  
  
  
numberList = [number for number in range(1,11)]  
print(numberList)  
#1.1 使用for循环来实现列表的奇数过滤  
numberOddList = []  
for number in numberList:  
 if (number%2 == 1):  
 numberOddList.append(number)  
print(numberOddList)  
  
#1.2 使用filter函数来实现列表的奇数过滤  
def isOdd(number):  
 return number % 2 == 1  
numberOddList= list(filter(isOdd, numberList))  
print(numberOddList)  
  
#1.3 使用filter函数与匿名函数相结合的奇数过滤  
numberOddList = list(filter(lambda number : number%2 == 1, numberList))  
print(numberOddList)  
  
#1.4 不能直接使用类表推到式的处理  
numberOddList = [number%2 == 1 for number in numberList]  
print(numberOddList)  
  
elementList = ['A', '', 'B', 'd ', 'C', ' good']  
#使用filter和map函数结合对字符串进行清洗  
def elementStrip(element):  
 return element.strip()  
  
def isEmpty(element):  
 if len(element) == 0:  
 return False  
 return True  
  
elementsStripList = list(map(elementStrip, elementList))  
elementsCleanList = list(filter(isEmpty, elementsStripList))  
print(elementsCleanList)  
  
  
#使用filter函数、map函数与匿名函数的例子  
elementsStripList = list(map(lambda element:element.strip(), elementList))  
print(elementsStripList)  
elementsCleanList = list(filter(lambda element: len(element) != 0, elementsStripList))  
print(elementsCleanList)  
  
  
#不推荐的写法，该写法会导致可读性极差  
elementsCleanList = list(filter(lambda element: len(element) != 0, list(map(lambda element:element.strip(), elementList))))  
print(elementsCleanList)

### 4.1.8 067高级函数的操作(30 reduce)

#reduce的使用：reduce(function, iterable[, initializer])  
#reduce()函数把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算。  
  
from functools import reduce  
#1.1 使用reduce函数典型代码  
def add(numberOne, numberTwo):  
 return numberOne + numberTwo  
  
def muliply(numberOne, numberTwo):  
 return numberOne \* numberTwo  
  
numberList = [number for number in range(1,6)]  
print(numberList)  
  
numberAdd = reduce(add, numberList)  
print(numberAdd)  
  
numberMul = reduce(muliply, numberList)  
print(numberMul)  
  
#1.2 使用reduce函数与匿名函数的结合  
numberList = [number for number in range(1,11)]  
print(numberList)  
  
#使用reduce函数对单个列表的相加操作  
numberAdd = reduce( lambda numberOne, numberTwo : numberOne + numberTwo , numberList)   
print(numberAdd)  
  
#使用reduce函数对单个列表的相乘操作  
numberMul = reduce( lambda numberOne, numberTwo : numberOne \* numberTwo , numberList)   
print(numberMul)

### 4.1.9 068高级函数的操作(40 sorted)

#使用sorted函数的典型用法  
numbers = [5,7,-6,3,4,1,2,-60]  
numbersSorted = sorted(numbers)  
print(numbersSorted)  
numbersSorted = sorted(numbers,reverse = True)  
print(numbersSorted)  
  
strings = ['cake','apple','Good','book','test','Air']  
stringsSorted= sorted(strings)  
print(stringsSorted)  
stringsSorted= sorted(strings, reverse = True)  
print(stringsSorted)  
  
#使用key参数的sorted函数用法  
#sorted()函数也是一个高阶函数，它还可以接收一个key函数来实现自定义的排序  
numbers = [5,7,-6,3,4,1,2,-60]  
numbersSorted = sorted(numbers, key = abs)  
print(numbersSorted)  
numbersSorted = sorted(numbers , key = abs, reverse = True)  
print(numbersSorted)  
  
strings = ['cake','apple','Good','book','test','Air']  
stringsSorted= sorted(strings, key = str.lower)  
print(stringsSorted)  
stringsSorted= sorted(strings, key = str.lower, reverse = True)  
print(stringsSorted)

### 4.1.10 070生成器(Generator)的使用

#1.1 生成器(generator)和列表推导式(comprehensions)的差异  
listFromComprehension = [x\*x for x in range(10)] #使用[]代表列表推导式  
print(listFromComprehension)  
  
#如果列表元素可以按照某种算法推算出来，那我们是否可以在循环的过程中不断推算出后续的元素呢？  
#这样就不必创建完整的list，从而节省大量的空间。  
listFromGenerator = (x\*x for x in range(10)) #使用()代表生成器  
for element in listFromGenerator:  
 print(element)  
  
#1.2 斐波拉契数列(Fibonacci)的实现  
  
#传统实现  
def Fibonacci(max):  
 count, a, b = 0, 0, 1  
 while count < max:  
 print(b)  
 a, b = b, a+b  
 count = count + 1  
Fibonacci(6)  
  
#生成器实现  
def Fibonacci():  
 a, b = 0, 1  
 while True:  
 yield b  
 a, b = b, a+b  
fib = Fibonacci()  
listFib = [next(fib) for i in range (6)]  
print(listFib)

### 4.1.11 tempCodeRunnerFile

#1.5 对多个列表的操作的传统方式与使用map方式的比较  
numberOneList = [number for number in range(1,11)]  
numberTwolist = list( map( lambda number : round(number \* 0.1, 1), numberOneList ) ) #python是以双精度(64)位来保存浮点数，多余的位会被截掉，不使用round看到的是0.1  
numberThreeList = list( map( lambda number : number \* 10, numberOneList ) )  
  
#对多个列表相同位置元素的相加操作的传统方式  
numberSumList = []  
for index in (range(len(numberOneList))): #使用列表长度来进行索引的控制  
 numberSum = numberOneList[index] + numberTwolist[index] + numberThreeList[index]  
 numberList.append(numberSum)  
print(numberSumList)  
  
#使用map函数对多个列表相同位置元素的相加操作  
numberSumList = list(map( lambda numberOne, numberTwo, numberThree : numberOne + numberTwo + numberThree, numberOneList, numberTwolist, numberThreeList ))  
print(numberSumList)