目录

1 基础部分

1.1 list

1.1.1 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)

# 数值列表的定义
ages = [1,2,3,4,5,6,7,8,9,10]
print(ages)
```

1.1.2 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')
```

1.1.3 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']
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)
```

1.1.4 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)
```

1.1.5 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)
```

1.1.6 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)
```

1.1.7 list(数值列表)

```
# range 的含义
for value in range(1,10):
    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))
```

1.1.8 list(列表推导)

```
# 列表推导
numbers = [number for number in range(1,101)] #列表的构造器
print(numbers)

#带条件的列表推导[表达式 for 变量 in 列表 if 条件]
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
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(evenNumbers)
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)

#带计算的列表推导式[表达式 for 变量 in 列表]
```

```
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)

#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)
```

1.1.9 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)
```

1.2 set

1.2.1 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)
```

1.2.2 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)
```

1.3 dictionary

1.3.1 dictionary(字典定义)

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

1.3.2 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))
```

1.3.3 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。')
```

1.3.4 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)
```

1.3.5 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))
```

1.3.6 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])
print(nameScoreDic)

#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']
nameScores = [(name,score) for name in names for score in scores]
print(nameScores)
```

1.4 tuple

1.4.1 tuple(元组定义)

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

1.4.2 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)
```

1.4.3 tuple(元组的访问)

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

1.4.4 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
```

1.5 while

1.5.1 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)
```

1.5.2 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')
```

1.5.3 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)
```

1.5.4 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')
```

1.6 if-else

1.6.1 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())
```

1.6.2 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())
```

1.6.3 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
```

1.6.4 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')
```

1.6.5 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')
```

1.7 function

1.7.1 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')
```

1.7.2 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')
```

1.7.3 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')
```

1.7.4 function(函数的返回值)

```
# 函数返回值
def get_fullName(firstName,lastName):
    return (firstName+lastName)

fullName = get_fullName('张','三')
print(fullName)

def get_name(firstName,lastName,middleName = ''):#使用默认值增强灵活性
    return(firstName + middleName + lastName)

fullName = get_name('张','三')
print(fullName)
fullName = get_name('张','三','贵')
print(fullName)
```

1.7.5 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')

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')
```

1.7.6 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)
```

1.7.7 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)

1.7.8 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)

1.7.9 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)
```

1.7.10 function(函数作为参数)

```
# 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**2 + width**2

def volumeCuboid(length,width,height): #长方体体积
    return length**3 + width**3 + height**3

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))
```

1.7.11 function(函数作为返回值)

```
# 函数作为返回值计算几何特性
def calculateGeometry(calculateFor):
    if calculateFor == 'aeraRectangle':
        def aeraRectangle(length,width): #矩形面积
            return length**2 + width**2
        return aeraRectangle #函数作为返回值
    if calculateFor == 'volumeCuboid':
        def volumeCuboid(length,width,height): #长方体体积
            return length**3 + width**3 + height**3
        return volumeCuboid #函数作为返回值

calGeometry = calculateGeometry('volumeCuboid')
print(calGeometry(1,2,3))
calGeometry = calculateGeometry('aeraRectangle')
print(calGeometry(1,2))
```

1.7.12 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)))
```

1.7.13 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
```

1.8 class

1.8.1 class(类的基本定义)

```
# 1.1 类的基本定义和引用
class People():
    def __init__(self,name,age):
        self.name = name #类的属性定义
        self.age = age
    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
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)

people01.name = '李晓红' #对象的属性改变
people01.study()
print( 'name 属性的值：' + people01.name)
```

1.8.2 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)
```

1.8.3 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)
```

1.8.4 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):
        def __init__(self,name,age):
                super().__init__(name,age)

student = Student('李莉',22)
student.getPeopleInfo()
```

1.8.5 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()
```

1.8.6 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 进阶部分

2.1 exception

2.1.1 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) )
```

2.1.2 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) )
```

2.2 fileRead

2.2.1 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)
```

2.2.2 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)
```

2.2.3 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）.
```

2.2.4 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
```

2.2.5 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())
```

2.2.6 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')
```

2.2.7 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
```

2.3 fileWrite

2.3.1 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')
```

2.3.2 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'
```

2.3.3 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())
```

2.4 wordsAnalysis

2.4.1 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))
```

2.4.2 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('”  ','').replac
e('  “','').replace('?','').replace(';','').lower()
        wordsProcessed.append(wordProcessed)
print(words)
print(len(words))
```

2.4.3 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('”  ','').replac
e('  “','').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))
```

2.4.4 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)
```

2.4.5 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)
```

2.4.6 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
```

2.5 lamdaExpression

2.5.1 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))
```

2.5.2 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)

2.5.3 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

2.5.4 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))
```

2.5.5 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)
```

2.6 regularExpresssion

2.6.1 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)
```

2.6.2 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)
```

2.6.3 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)
```

2.6.4 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！！！')
```

2.7 dataVisualization

2.7.1 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()
```

2.7.2 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()
```

2.7.3 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
```

2.7.4 dataVisualization(pyecharts 柱状图词频表达)

```
#2.2 pyecharts 对分词结果的处理
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.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/220pyechartsBarWordsAna.html')   # 生成本地 HTML 文件
```

2.7.5 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
```

2.7.6 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
```

2.7.7 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')
```

2.7.8 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'))
```

2.7.9 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 高级部分

3.1 语法最佳实践

3.1.1 010 string 拼接最佳操作

```
#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)
```

3.1.2 020 enumerate 函数使用

```
#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)
```

3.1.3 030 zip 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)
```

3.1.4 040 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)
```

3.1.5 050 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),None) for number in range(5))
print(squares.keys())
```

3.1.6 060 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。
```

3.1.7 070 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)
```