上海大学

SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号	第 2 组
实验序号	3
学　　号	21120860
姓　　名	刘虚谷
日　　期	2023 年 5 月 1 日

Python 计算实验报告撰写提纲

一、实习目的与要求

1. 熟悉 Python 的函数定义；
2. 熟悉 Python 的面向对象定义；
3. 掌握 Python 语言基本语法；

二、实习环境

1. 操作系统不限；
2. Python IDLE、PyCharm 等开发环境；

三、实习内容

6．阅读和运行 grid.py (支持 Python2.x，请修改代码使得 Python3.x 下同样可以正常运行)，分析 grid 所实现功能中各个函数之间的调用关系，绘制这种调用关系的流程图（可用 Visio 等软件绘制，流程图放到实验报告中）。比较 grid.py 的实现方法和 3（1）中你所实现的绘制表格函数的差异，并且把学习体会写在实验报告中。

7．阅读和运行 myArray.py 和 myMatrix.py，分析其中类的功能，比较类的定义中同名函数实现上的差异，并写入实验报告。

8、阅读和运行 Kangaroo.py，调用和测试其种所定义的类 Kangaroo 的方法，分析方法实现中的 bug，修正，写入实验报告。

（二）设计实验（小组验收，代码提交，算法设计和测试写入实验报告）

1. 函数和数据结构复习

（1）编写 Ackermann 函数的递归实现 Ack（m,n）

测试 Ack（

3，4）的值，阅读 https://en.wikipedia.org/wiki/Ackermann_function，

分析 m 和 n 取值对函数值计算的影响，深入理解递归。

（

2）编写一个函数，实现从序列中移除重复项，且保持元素间顺序不变。

生成随机的列表和字典，验证所实现函数的功能。

2．编写拥有 a、对象成员 hour，minute 和 second 的时间类 Time；b、重载__str__和__add__方法；c、方法 time2int：把时间对象转换为秒数；d、方法 printtime：输出时间；

e、方法 isafter：判断两个时间对象的先后；

f、方法 increment：

计算对象经过 n〉0 秒后时间；g、方法 isvalid：判断时间对象合法性。在主函数设计代码验证 Time 各个方法的正确性。

3．马尔可夫文本分析和应用（1）马尔可夫文本分析计算文本中单词组合和其后续单词（含标点符号）

的映射，这个单词组合被称为马尔可夫分析的前缀，前缀中单词的个数被称为马尔可夫分析的“阶数”。编写 Python 代码实现某个文本的 n 阶马尔可夫文本分析，并且将分析结果记录在字典中。

（
2）采用（1）所实现的马尔可夫分析模块，对“emma.txt”或“whitefang.txt”进行马尔可夫分析，运用 n 阶马尔可夫分析的结果生成由 m 个句子（注意首字母大写和结尾标点符号）组成的随机文本。分析所生成文本的语义自然性和阶数 n 的关系。
（
3）尝试采用 Python 不同的序列数据结构表示前缀，比较运行效率的差异。
4．模拟快餐订餐场景
（1）定义 4 个类：Customer 顾客类，Employee 商户类，Food 食物类 以及 Lunch 订餐管理。
（
2）Lunch 类包含 Customer 和 Employee 实例，具有下单 order 方法，该方法要求 Customer 实例调用自身的 placeOrder 向 Employee 对象要求下单，并且获得 Employee 对象调用 takeOrder 生成和返回一个 Food 对象，Food 对象应当包含了食物名字符串。调用关系如下：
Lunch.order—〉Customer.placeOrder—〉Employee.takeOrder—〉Food
（
3）Lunch 类包含 result 方法，要求 Customer 打印所收到的食物订单。
（
4）编写交互式界面验证所设计的订餐系统。
5．编制系列单词处理函数，分别实现下述功能，并设计测试用例验证程序的正确性，请在实验报告中说明所使用的正则表达式。
（1）编写函数 rotateword，接收一个字符串 strsrc 以及一个整数 n 作为参数，返回新字符串 strdes，其各个字母是 strsrc 中对应位置各个字母在字母表中“轮转”n 字符后得到的编码。
（
2）编写函数 avoids，接收一个单词和一个含有禁止字母的字符串，判断该单词是否含有禁止字母。
（
3）编写函数 useonly，接收一个单词和一个含有允许字母的字符串，判断该单词是否仅仅由允许字母组成。
（
4）编写函数 useall，接收一个单词和一个含有需要字母的字符串，判断该单词是否包含了所有需要字母至少一个，并输出 words.txt 中使用了所有元音字母 aeiou 的单词。
（
5）编写函数 hasnoe，判断一个英语单词是否包含字母 e，并计算 words.txt 中不含 e 的单词在整个字母表中的百分比。
（
6）编写函数 isabecedarian，判断一个英语单词中的字母是否符合字母表序，并且输出 words.txt 中所有这样的单词。

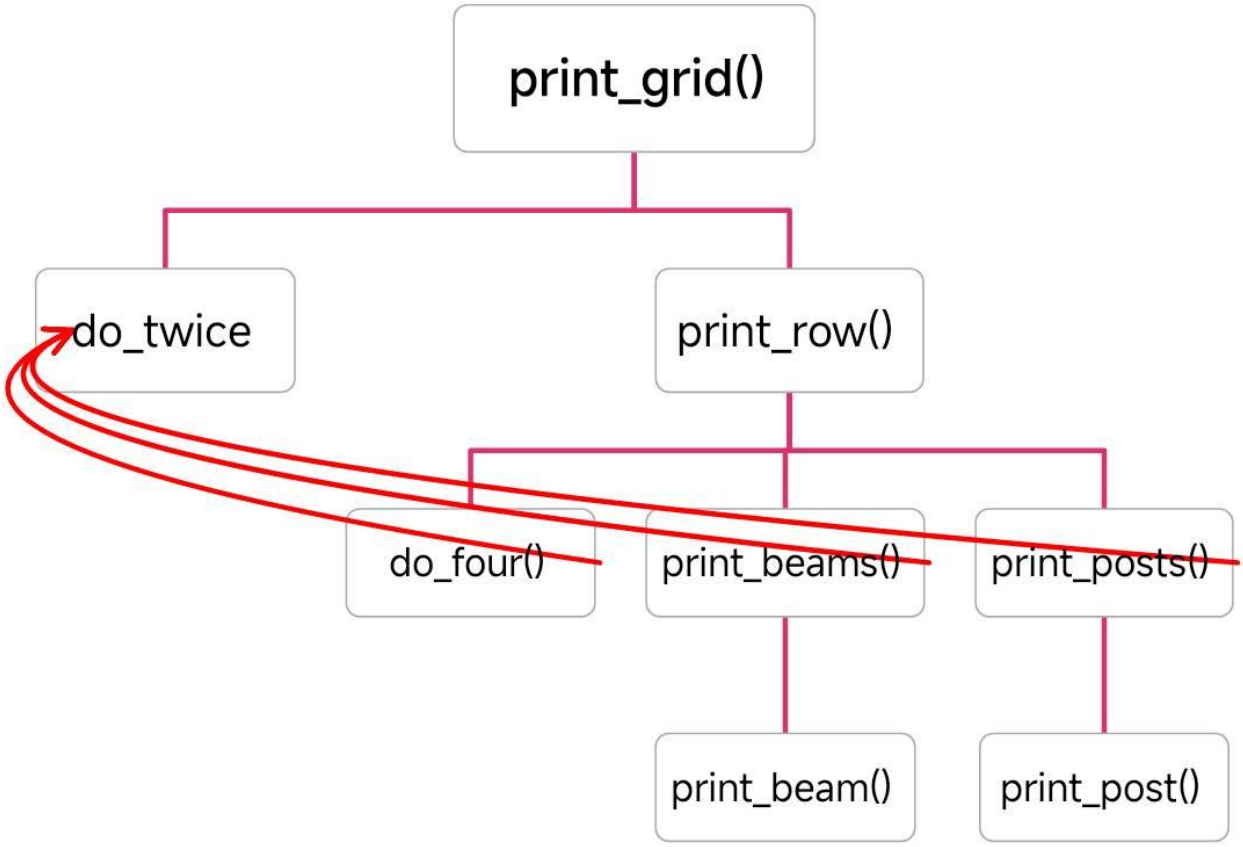
四、实习内容的设计与实现

6.

```
def do_twice(f):
    f()
    f()

def do_four(f):
    do_twice(f)
    do_twice(f)

def print_beam():
    print('+ - - - -',end='')

def print_post():
    print('|         ',end='')

def print_beams():
    do_twice(print_beam)
    print('+',end='\n')

def print_posts():
    do_twice(print_post)
    print('|',end='\n')

def print_row():
    print_beams()
    do_four(print_posts)

def print_grid():
    do_twice(print_row)
    print_beams()

print_grid()


# here is a less-straightforward solution to the
# four-by-four grid

def one_four_one(f, g, h):
    f()
    do_four(g)
    h()
```

```
def print_plus():
    print('+',end='')

def print_dash():
    print('-',end='')

def print_bar():
    print('|',end='')

def print_space():
    print(' ',end='')

def print_end():
    print('\n')

def nothing():
    "do nothing"
```

Python2.x 中可在 print 后加,这样两个 print 可连续输出，Python3.x 中经测试是不行的，而是在 print 设置 end=’’,这样可达到同样的效果。



7.

我认为两个类同名函数功能实现有三个主要区别，举以下代码作比较：

myArray.py 中：

```
 def __sub__(self, n):
        if not self.__IsNumber(n):
            print('- operating with ', type(n), ' and number type is not supported.')
            return
        b = MyArray()
        b.__value = [item-n for item in self.__value]
        return b
def __mul__(self, n):
        if not self.__IsNumber(n):
            print('* operating with ', type(n), ' and number type is not supported.')
            return
        b = MyArray()
        b.__value = [item*n for item in self.__value]
        return b
   def __str__(self):
        return str(self.__value)
```

myMatrix 中：

```
def __sub__(self, n):
        #参数是整数或实数，则返回矩阵
        #其中的每个元素为原矩阵中元素与该整数或实数的加法结果
        if type(n) in (int, float, complex):
            return self.__operate(n, '-')
        elif isinstance(n, simNumpyArray):
            #如果参数为同类型矩阵，且大小一致，则为两个矩阵中对应元素相减
            if n.__row==self.__row and n.__col==self.__col:
                #先实例化一个临时对象，其值临时为[1]
                return self.__matrixAddSub(n, '-')
            else:
                print('two matrix must be the same size')
                return
        else:
            print('data type error')
            Return
def __mul__(self, n):
        #参数是整数或实数，则返回矩阵
        #其中的每个元素为原矩阵中元素与该整数或实数的加法结果
        if type(n) in (int, float, complex):
```

```
            return self.__operate(n, '*')
        elif isinstance(n, simNumpyArray):
            #如果参数为同类型矩阵，且第一个矩阵的列数等于第二个矩阵的行数
            if n.__row==self.__col:
                data = []
                for row in self.__data:
                    t = []
                    for ii in range(n.__col):
                        col = [c[ii] for c in n.__data]
                        tt = sum([i*j for i,j in zip(row,col)])
                        t.append(tt)
                    data.append(t)
                c = simNumpyArray([t for i in data for t in i])
                c.reshape((self.__row, n.__col))
                return c
            else:
                print('size error.')
                return
        else:
            print('data type error')
            Return

def __str__(self):
        return '\n'.join(map(str, self.__data))

def __operate(self, n, op):
        b = simNumpyArray([t for i in self.__data for t in i])
        b.reshape((self.__row, self.__col))
        b.__data = [[eval(str(j)+op+str(n)) for j in item] for item in b.__data]
        return b

def __matrixAddSub(self, n, op):
        c = simNumpyArray([1])
        c.__row = self.__row
        c.__col = self.__col
        c.__data = [[eval(str(x[i])+op+str(y[i])) for i in range(len(x))] for x,y in zip(self.__data,
n.__data)]
        return c
```

1. 以两者的__sub__函数为例：myarray 中__sub__函数用来完成数组的所有元素与一个数的减法，实现由一个列表推导式完成，较简单。在 myMatrix 中__sub__函数不仅用来完成矩阵与一个数相减，还用来完成矩阵与矩阵间的减法，而矩阵与一个数的减法就调用一个__operate 通用函数实现，而__operate 则通过字符串以及相应的 eval 函数完成它的通用功能，而矩阵间的相减则主要通过列表推导式加上 zip 函数实现两矩阵间相应元素的减法。
2. 以两者间的__mul__函数为例：myarray 中的__mul__函数同样较简单，我们主要看看

myMatrix 中__mul__函数的实现，准确来说，是矩阵间相乘的实现，在满足其代码设定的条件下，我们用了一个 for 循环锁定 self 中的某一行，之后又用一个 for 循环与前一个 for 循环锁定了 n 中的某一列,然后列表推导式与 zip 函数结合算出新矩阵中对应的某一个元素的值存入列表，最后根据矩阵相乘的性质调用 reshape()函数，完成最后的位置的调整。
3. 以两者中的__str__函数为例：myarray 中的__str__函数直接发挥相应的数组字符串形式，myMatrix 中的__str__函数则通过 map 函数进行对应的分割然后加上‘\n’达到每个列表字符串一行的效果。

8.

```
def __init__(self, contents=None):
        """initialize the pouch contents; the default value is
        an empty list"""
        if contents==None:
            contents=[]
        self.pouch_contents = contents
```

这是修改后的__init__函数,之前的函数默认参数为[]，但由于默认值参数只在函数定义时进行解释，然后默认值参数的引用不在变化。而对于列表，字典这样可变类型的默认值参数，这一点可能会导致很严重的错误。

二．设计实验

1.（1）：

```
def ack(m,n):
    if m==0:
        y=n+1
        return y
    elif m>0 and n==0:
        y=ack(m-1,1)
        return y
    elif m>0 and n>0:
        y=ack(m-1,ack(m,n-1))
        return y

print('ack(3,4)的值为：',ack(3,4))
```

根据题意，很容易写出递归函数，最后得出值为 125，该递归函数值受 m,n 取值影响很大，当 m,n 分别取值为 4,5 时，就发生了栈溢出。

1.（2）：

```
import random
def dedupe(list):
    seen = []
    for i in list:
        if i not in seen:
            seen.append(i)
    return seen
a = [random.randint(1,10) for i in range(20)]
print('原列表',a,'\n')
print('去重后',dedupe(a),'\n\n')

def dedupe(items,key=None):
    seen=set()
    for item in items:
        value=item if key is None else key(item) #不可哈希
        if value not in seen:
            yield item    #生成器保证顺序不变
            seen.add(value)

a=[
    {'a':1,'b':2,'c':3},
    {'a':1,'b':3,'c':3},
    {'a':1,'b':4,'c':3},
    {'a':1,'b':2,'c':3},
    {'a':1,'b':3,'c':3},
    {'a':1,'b':1,'c':3},

]
print('原字典',a,'\n')
print('去重后',list(dedupe(a,key=lambda d:(d['a'],d['b'],d['c']))))
```

若序列元素可哈希，则用第一个 dedupe 函数，利用 in 判断是否重复，若序列元素不可哈希，则利用第二个 dedupe 函数,利用传递给 key 参数的函数将不可哈希的元素转换为可哈希的类型,在进行重复元素的判断,在这两个函数中都运用了生成器(yield)，即可以保存元素，又能保持元素间顺序不变，最后返回生成器对象并转化为 list 查看。

2：

```
class myTime():
    def isvalid(self,h,m,s):
```

```
        if h not in range(0,24) or not isinstance(h,int):
                print('hour must be a number and be included in [0-23]')
                return False
        elif m not in range(0,60) or not isinstance(m,int):
                 print('minute must be a number and be included in [0-59]')
                 return False
        elif s not in range(0,60) or not isinstance(s,int):
                 print('second must be a number and be included in [0-59]')
                 return False
        return True

def __init__(self,h=0,m=0,s=0):
        if self.isvalid(h,m,s):
                self.hour=h
                self.minute=m
                self.second=s
        else:
                return
def __str__(self):
        return str(self.hour)+':'+str(self.minute)+':'+str(self.second)
def __add__(self,n):
        m=myTime()
        m.hour=self.hour
        m.minute=self.minute
        m.second=self.second
        if isinstance(n,int):
                m.second+=n
                m.minute+=m.second//60
                m.second%=60
                m.hour+=m.minute//60
                m.minute%=60
                m.hour%=24
        elif isinstance(n,myTime):
                m.second+=n.second
                m.minute+=(m.second//60+n.minute)
                m.second%=60
                m.hour+=(m.minute//60+n.hour)
                m.minute%=60
                m.hour%=24
        return m
def time2int(self):
        m=self.hour*3600+self.minute*60+self.second
        return m
def printtime(self):
```

```
            print(self.hour,self.minute,self.second,sep=':')
        def isafter(self,m):
            if self.hour>m.hour:
                print('The latter precedes the former')
            elif self.hour<m.hour:
                print('The former precedes the latter')
            elif self.minute>m.minute:
                print('The latter precedes the former')
            elif self.minute<m.minute:
                print('The former precedes the latter')
            elif self.second>m.second:
                print('The latter precedes the former')
            elif self.second<m.second:
                print('The former precedes the latter')
        def increment(self,n):
            m=self+n
            m.printtime()
if __name__=='__main__':
  time1=myTime(0,0,0)
  time2=myTime(23,59,59)
  print('time1 为:',time1,sep='')
  print('time2 为:',end='')
  time2.printtime()
  time1.second=2
  print('time1+time2 为:',time1+time2,sep='')
  print('time2 对象转化为秒数为:',time2.time2int(),sep='')
  print('time2 加两秒后为:',end='')
  time2.increment(2)
  print('比较 time1 和 time2 的先后',end=''),
  time1.isafter(time2)
```

定义 mytime 类时，先定义了 isvalid 函数用来判断时分秒是否为数字且在规定范围内，然后是__init__函数用来初始化对象变量，然后是对__str__和__add__的一个重载，使得 print 和+时，分别输出时：分：秒和返回 mytime 类对象。Isafter 函数按时分秒的优先级依次进行比较，

Increment 函数则直接利用重载的+函数。最后测试成功。

3.（1）：

```
def markfuemphasize(filename,n):
```

```
    file=open(filename,'r',encoding='UTF-8')
    words=file.read().split()
    d={}
    for i in range(len(words)-n):
        s=tuple(words[i:i+n:1])
        if s in d:
            d[s].append(words[i+n])
        else:
            d[s]=[words[i+n]]
    return d
```

将 filename 文本的 n 阶分析结果放入字典并返回字典 d,字典的键为 n 个单词组成的元组，值为列表，列表中的元素为对应键后的单词，可重复。

3.（2）：

```
def markfumakefile(d,n,m):
    ''' 通过 markfuemphasize()函数返回的字典生成随机的 m 个句子'''
    import random
    firstword=random.choice(list(d))
    while firstword[0][0].islower():
        firstword=random.choice(list(d))
    chain=list(firstword)
    l=0
    #用 l 记录已生成的句子数
    for i in chain:
        if i[-1] in ('.','。','!','?'):
            l+=1
    while l<m:
      words=chain[-n:-1:1]
      #这里共有 n-1 个元素
      words.append(chain[-1])
      #chain[-1]为 chain 中最后一个元素
      word=random.choice(d[tuple(words)])
      chain.append(word)
      if word[-1] in ('.','?','!','。'):
            l+=1
    mphrase=' '.join(chain)
    return mphrase

if __name__=='__main__':
    import time

    start = time.time()
```

```
d1 = markfuemphasize('emma.txt', 5)
end = time.time()
print(end - start, end='\n')
start = time.time()
mphrase1 = markfumakefile(d1, 5, 10)
end = time.time()
print(end - start, end='\n')
print(mphrase1, end='\n')
start = time.time()
d2 = markfuemphasize('whitefang.txt', 5)
end = time.time()
print(end - start, end='\n')
start = time.time()
mphrase2 = markfumakefile(d2, 5, 10)
end = time.time()
print(end - start, end='\n')
print(mphrase2, end='\n')
```

之后我们编写了利用 n 阶马尔科夫分析得出的字典来随机生成一个由 m 个句子组成的自然语言文本，先在字典中随机选取一个键，既由 n 个单词组成的句子，且所选取的句子首字母大写，然后用变量 l 记录句子中含有的.。？！这 4 个标点符号依次表示以生成的句子数，当仍没有生成 m 个句子时，利用切片找出键，并在对应的列表中随机选择一个单词，并判断是否含有上面 4 个标点符号，最后返回生成的字符串，即生成的文本。

3.（3）：

```
def markfuemphasize(filename,n):
    ''' 将 filename 文本的 n 阶分析结果放入字典并返回字典'''
    file=open(filename,'r',encoding='UTF-8')
    words=file.read().split()
    d={}
    for i in range(len(words)-n):
        s=' '.join(words[i:i+n:1])
        if s in d:
            d[s].append(words[i+n])
        else:
            d[s]=[words[i+n]]
    return d
```

```
def markfumakefile(d,n,m):
    ''' 通过 markfuemphasize()函数返回的字典生成随机的 m 个句子'''
    import random
    firstword=random.choice(list(d))
    while firstword[0].islower():
        firstword=random.choice(list(d))
    chain=firstword.split()
    l=0
    #用 l 记录已生成的句子数
    for i in chain:
        if i[-1] in ('.','。','!','?'):
            l+=1
    while l<m:
      words=chain[-n:-1:1]
      #这里共有 n-1 个元素
      words.append(chain[-1])
      #chain[-1]为 chain 中最后一个元素
      words=' '.join(words)
      word=random.choice(d[words])
      chain.append(word)
      if word[-1] in ('.','?','!','。'):
            l+=1
    mphrase=' '.join(chain)
    return mphrase

if __name__=='__main__':
    import time
    start=time.time()
    d1=markfuemphasize('emma.txt',35)
    end = time.time()
    print(end - start,end='\n')
    start = time.time()
    mphrase1=markfumakefile(d1,35,10)
    end=time.time()
    print(end-start,end='\n')
    print(mphrase1, end='\n')
    start=time.time()
    d2=markfuemphasize('whitefang.txt',35)
    end = time.time()
    print(end - start, end='\n')
    start = time.time()
    mphrase2=markfumakefile(d2,35,10)
    end=time.time()
    print(end - start,end='\n')
```

```
    print(mphrase2,end='\n')
```

这一函数则利用了字符串作为字典的键，其他基本类似，最后测试发现利用字符串比利用元组的运行效率整体要慢些。

4.：

```
class Customer:
    def __init__(self, name):
        self.name = name
        self.foods = []
    def placeOrder(self, employee):
        while True:
            choice = input("请问您要点什么餐？(按 n 结束点餐) ")
            if choice == 'n':
                break
            food = employee.takeOrder(choice)
            self.foods.append(food)
    def showOrder(self):
        print(f"{self.name} 点了以下餐品：")
        for food in self.foods:
            print(food.name, end=" | ")
        print()

class Employee:
    def __init__(self,name):
        self.name=name
    def takeOrder(self, choice):
        food = Food(choice)
        print(f"{food.name} 点餐成功！")
        return food

class Food:
    def __init__(self, name):
        self.name = name

class Lunch:
    def __init__(self, customer, employee):
        self.customer = customer
        self.employee = employee
    def order(self):
        self.customer.placeOrder(self.employee)
    def result(self):
        self.customer.showOrder()

customer = Customer("汤姆")
```

```
employee = Employee("肯德基")
lunch = Lunch(customer, employee)

while True:
    print("请选择您要进行的操作：")
    print("1. 我要点餐")
    print("2. 展示菜单")
    print("3. 退出程序")

    choice = input()

    if choice == "1":
        lunch.order()
    elif choice == "2":
        lunch.result()
    elif choice == "3":
        break
    else:
        print("输入有误，请重新输入。")
```

定义 了 4 个类：Customer 顾客类，Employee 商户类，Food 食物类 以及 Lunch 订餐管理，并且 Lunch 类包含 Customer 和 Employee 实例，具有下单 order 方法， Customer 实例则调用自身的 placeOrder 向 Employee 对象要求下单，并且获 得 Employee 对象调用 takeOrder 生成和返回一个 Food 对象，Food 对象应当包含 了食物名字符串。Lunch 类里定义了 result 方法，能够打印 Customer 所收到的食物订单，最后 编写了一个简单的交互式界面验证所设计的订餐系统。

5. （1）：

```
#def rotateword(strsrc,n):
#       import string as s
#       lower=s.ascii_lowercase[n:]+s.ascii_lowercase[:n]
#       upper=s.ascii_uppercase[n:]+s.ascii_uppercase[:n]
#       table=''.maketrans(s.ascii_letters,lower+upper)
#       strdes=strsrc.translate(table)
#       return strdes
```

```
def rotateword(strsrc, n):
    alphabet = 'abcdefghijklmnopqrstuvwxyz' #  单词表
    regex = re.compile('[a-zA-Z]')
    strdes = ''
    for char in strsrc:
        if regex.match(char):
            is_upper = char.isupper()     #判断是否是大写
            char = char.lower()
            idx = (alphabet.index(char) + n) % 26     #轮转
            rotated_char = alphabet[idx]
            if is_upper:
                rotated_char = rotated_char.upper() #若为大写，重新大写
            strdes += rotated_char
        else:
            strdes += char
    return strdes
```

第一个 rotateword 函数利用了 maketrans 建立某个字母与其轮换 n 次后对应的映射表,translate 则将对应的字符串根据 maketrans 建立的映射表进行转换并返回。

第二个 rotateword 函数则主要利用了正则表达式，并记录某个字母是否大写，在进行轮换，之后根据之前对字母大小写的判断在进行转换，最后把它再加入字符串中。

5.（2）：

```
def avoids(word, forbidden):
    pattern = "[" + forbidden + "]" #  一组字符  [amk]  匹配  'a'，'m'或'k'
    if re.search(pattern, word):
        return True #  含有禁止字母
    return False #  不含禁止字母
```

利用正则表达式[禁止的字母]和 re 模块的 search 进行匹配。

6.（3）：

```
def useonly(word, allow):
  regex = re.compile('^[' + allow + ']+$') # ^  和  $  限制开头和结尾
  return bool(regex.match(word))
```

5.（4）：

```
def useall(word, allow):
    for c in allow:
```

```
            if not re.search(c, word):
                return False
        return True
def aeiou_words():
            list = set()
            with open('words.txt', 'r') as file:
                for word in file:
                    if re.search('[aeiou]', word) and useall(word.strip(), 'aeiou'):
                        # search 是否包含任何元音字母
                        list.add(word.strip())
            print(len(list),end='\n')
```

Aeiou_words 函数中 re.search 判断单词中是否含有元音字母，useall 函数用于判断是否含有所有元音字母，and 链接利用惰性求值提高查找效率。最后对 words.txt 分析满足条件的单词有 598 个。

5.（5）：

```
def hasnoe(char):
    return not re.search('e', char)

def noe_words():
    with open('words.txt', 'r') as file:
        sum = 0
        no_e = 0
        for word in file:
            sum += 1
            if hasnoe(word.strip()):
                no_e += 1
        percentage = (no_e / sum) * 100
        print("百分比为: {:.2f}%".format(percentage),no_e,end='\n')
```

利用 re.search 匹配含 e 的单词，noe_words 运行查找到满足条件的单词有 37641 个。

5.（6）：

```
def isabecedarian(word):
    return          bool(re.match(r'^a*b*c*d*e*f*g*h*i*j*k*l*m*n*o*p*q*r*s*t*u*v*w*x*y*z*$',
word))# re*  匹配 0 个或多个

def sort_words():
    list = set()
```

```
    with open('words.txt', 'r') as file:
        for word in file:
            if isabecedarian(word.strip()):
                list.add(word.strip())
    print(len(list),end='\n')
```

利用正则表达式和 re 模块的 search 函数

re.match(r'^a*b*c*d*e*f*g*h*i*j*k*l*m*n*o*p*q*r*s*t*u*v*w*x*y*z*$', word)其中*匹配前面字母的 0 次及以上的出现，sort_words 函数查找到满足要求的单词为 596 个。

```
if __name__ == '__main__':
    # 1
    assert rotateword('Hello, World!', 5) == 'Mjqqt, Btwqi!'
    assert rotateword('xyz123', 3) == 'abc123'
    assert rotateword('', 5) == ''
    # 2
    assert avoids("hello", "xyz") == False
    assert avoids("world", "xyz") == False
    assert avoids("Python", "pt") == True
    assert avoids("Java", "pt") == False
    # 3
    assert useonly('hello', 'helo') == True #  仅由 h,e,l,o 组成
    assert useonly('world', 'helo') == False
    assert useonly('hello!!', '!') == False
    # 4
    assert useall('hello!!', '!') == True
    aeiou_words()
    # 5
    assert hasnoe('hello!!') == False     #存在 e
    assert hasnoe('12ioo') == True          #不存在 e
    noe_words()
    # 6
    assert isabecedarian('a') == True
    assert isabecedarian('bdf') == True
    assert isabecedarian('eg') == True
    assert isabecedarian('ahy') == True
    assert isabecedarian('zxy') == False
    sort_words()
    print("测试通过")
```

利用 assert 断言，若运行无中止即运行成功。

五、测试用例

emma.txt

Whitefang.txt

Words.txt

六、收获与体会

在本次实验中，我们组首次上台与大家做了分享，这即锻炼了我们的代码能力，也锻炼了我们的演讲能力，同时在与别的组分享中，也认识到我们组的做的好的与不好的地方，让我们能够共同进步。