Python 计算实验报告

组　　号	第 8 组
实验序号	3
学　　号	19120191
姓　　名	汪雨卿
日　　期	2020 年　5 月　20 日

一、 实验目的与要求

1. 熟悉Python 的开发调试环境；
2. 熟悉Python 的流程控制;
3. 熟悉Python 的数据结构;
4. 掌握Python 语言基本语法;

二、 实验环境

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

三、 实验内容

1. 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。

2. 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。

3. 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。

4. 函数和数据结构复习

编写 Ackermann 函数的递归实现 Ack（m,n）

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > \end{cases}$$

a.测试 Ack(3,4)的值，阅读 https://en.wikipedia.org/wiki/Ackermann_function，分析 m 和 n 取值对涵数值计算的影响，深入理解递归。

b. 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

a. 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。

b. 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。

c. 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

a. 定义 4 个类：Customer 顾客类，Employee 商户类，Food 食物类 以及 Lunch 订餐管理。

b. 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

c. Lunch 类包含 result 方法，要求 Customer 打印所收到的食物订单。

d. 编写交互式界面验证所设计的订餐系统。

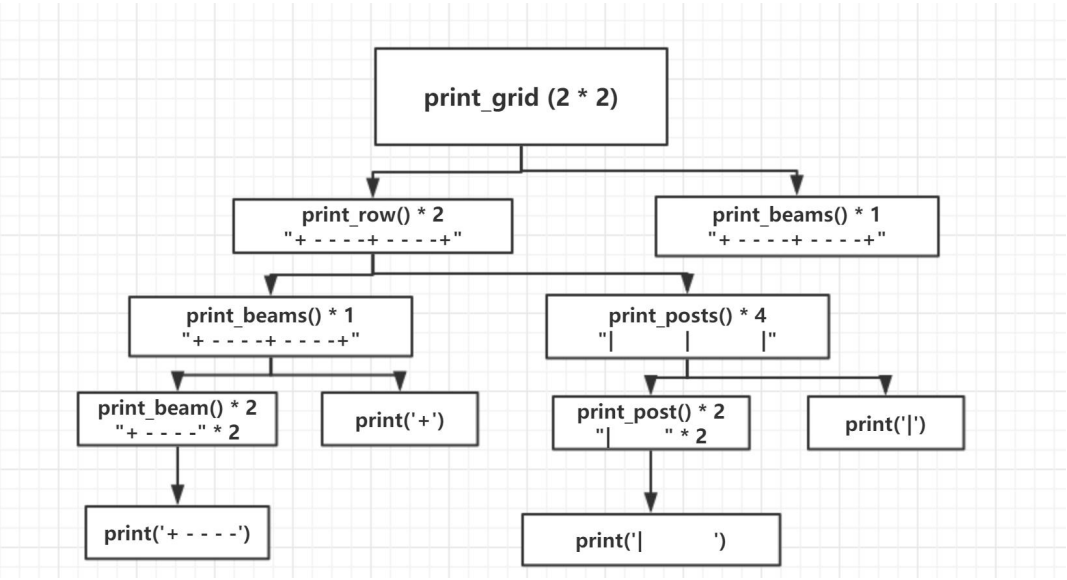
四、 实验内容的设计与实现

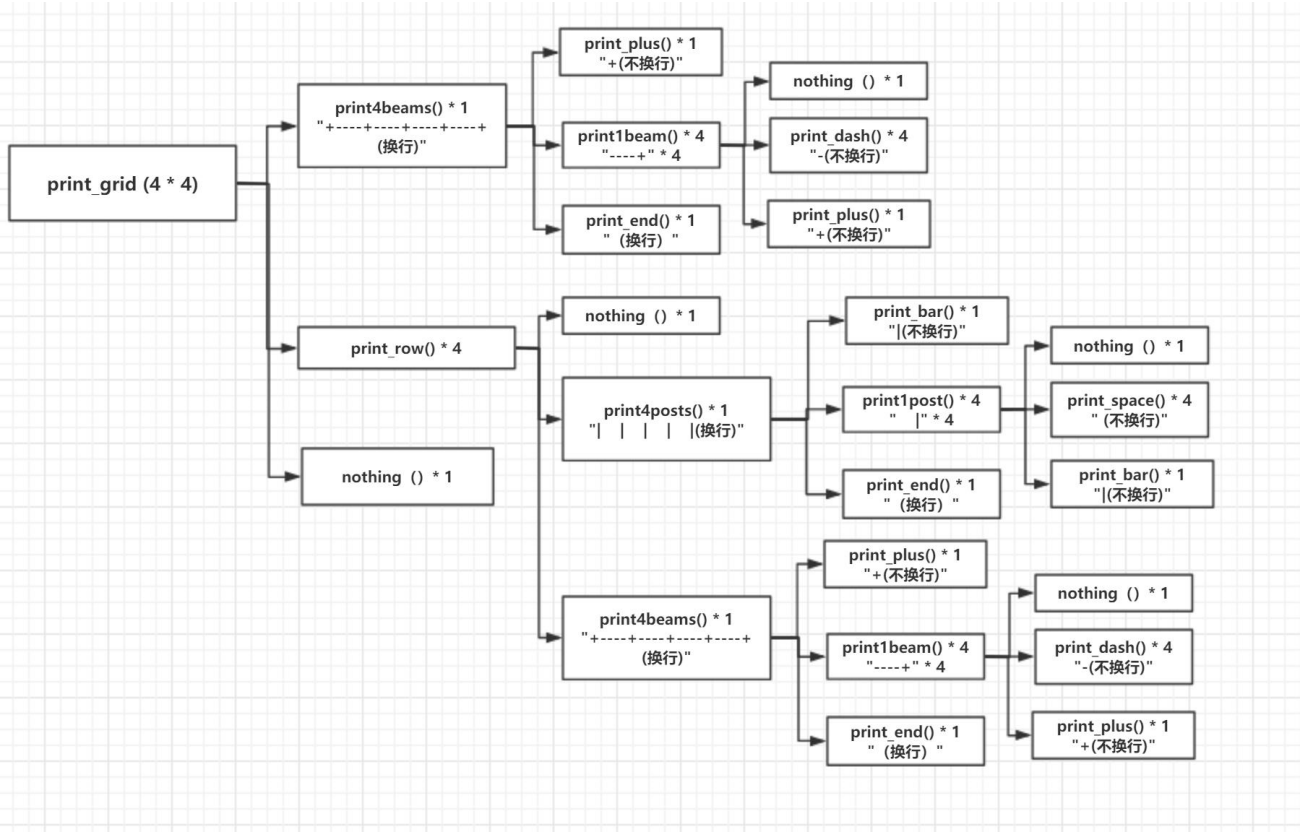
题目 1： 阅读和运行 grid.py（支持 Python2.x，请修改代码使得 Python3.x 下同样可以正常运行），分析 grid 所实现功能中各个函数之间的调用关系，绘制这种调用关系的流程图（可用 Visio 等软件绘制，流程图放到实验报告中）。比较 grid.py 的实现方法和 3（1）中你所实现的绘制表格函数的差异，并且把学习体会写在实验报告中。

【分析】

Grid 给出的代码实现的绘制网格的效果是通过将整个图片的每一划编写成一个函数，（也可以理解为一个工具）。然后，根据需求，编写函数组合所需要的笔画，不断的调用重复的工具，进而组合成最后的成果。

而我所写的函数所实现的功能，是通过一个模板函数，也是抓住了网格中存在做 1 次和做 4 次同样的内容的特点。可以通过需求，传入不同的符号，绘制所需要的图像。





【代码实现】

```
def tempLine(a, b):
    '''绘画半行模板'''
    print(a, end='')
    for i in range(4):
        print(b, end='')


def endPlusLine():
    """绘制加号后半行"""
    tempLine('+ ', '- ')
    print('+ ')


def endStraightLine():
    """绘制| 号后半行"""
    tempLine('| ', '  ')
    print('| ')


def plusLine():
    """绘制加号行"""
    tempLine('+ ', '- ')
    endPlusLine()


def straightLine():
    """绘制竖号行"""
    tempLine('| ', '  ')
    endStraightLine()


def my_grid():
    plusLine()
    for i in range(4):
        straightLine()
    plusLine()
    for i in range(4):
        straightLine()
    plusLine()


my_grid()
```

题目 2：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。

【类的功能】

myArray实现了+，-，*，/，//，%，**，len()长度，print（）输出，指定下标取值，根据下标修改值，in方法检测，==，<方法的重载，并且实现了模拟向量内积的功能。

myMatrix实现了修改大小，矩阵转置,计算三角函数sin, cos的功能，对于print()输出，+，-，*，真除法，//,**，==，<，in方法测试，__iter__迭代进行了重载。

【重名比较】

对于myMatrix中的数据需要更多的考虑行列的因素，再判断取数的时候都需要考虑行与列的影响。而对于myArray部分的数据都是一行多列的数组，进行相加的时候需要判断myArray的长度是否一致，若一致进行元素的操作。

题目 3：

3. 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。

【分析】

由于 Kangaroo 中的变量都没有设置成对象的变量，所以所声明的变量都是类的数据成员。因此及时没有给创建的对象 roo 新增元素，也都会指向之前加过元素的同一个对象。导致输出上的问题。

若要修改这个 bug,只需要将原来属于类的数据成员,修改成为每个对象的私有成员即可。

【代码实现】

```python
class Kangaroo(object):
    """a Kangaroo is a marsupial"""
    # 把[]改成None
    def __init__(self, contents=None):
        """initialize the pouch contents; the default value is
        an empty list"""
        if contents != None:
            # 把pouch_contents修改成__pouch_contents
            self.__pouch_contents = contents
        else:
            self.__pouch_contents = []

    def __str__(self):
        """return a string representaion of this Kangaroo and
        the contents of the pouch, with one item per line"""
        t = [object.__str__(self) + ' with pouch contents:']
        for obj in self.__pouch_contents:
            s = '    ' + object.__str__(obj)
            t.append(s)
        return '\n'.join(t)

    def put_in_pouch(self, item):
        """add a new item to the pouch contents"""
        self.__pouch_contents.append(item)
```

题目 4：函数和数据结构复习

编写 Ackermann 函数的递归实现 Ack（m,n）

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > \end{cases}$$

a.测试 Ack(3,4)的值，阅读 https://en.wikipedia.org/wiki/Ackermann_function，分析 m 和 n 取值对涵数值计算的影响，深入理解递归。

b. 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。

a.【实现原理】

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

从 Ackermann 函数的定义中可以看出，Ackermann 函数可以看成关于 n 的一个函数序列，其中第 0 个函数返回 n+1，而第 m 个函数则是将第 m-1 个函数对 1 迭代 n+1 遍。对较小的 m，该函数为：

Ackermann(0, n)=n+1

Ackermann(1, n)=n+2

Ackermann(2, n)=2*n+3

Ackermann(3, n)=2^(n+3)-3

Ackermann(4, n)=2^2^2^……^2-3，乘幂中共有 n+3 个 2。

【结果】

Ack（3，4） = 125

【代码实现】

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

b. **【实现原理】**

对于输入的内容，建立一个字典，将每次第一次出现的单词存入字典。通过增设 key 可以接受可哈希和不可哈希的数列。整个字典的生成是一个迭代器。

【结果】

对于列表的测试：

d [5, 4, 6, 5, 9, 3, 4, 2, 9, 4, 5, 9, 8, 5, 7, 4, 8, 7, 9, 1]

[5, 4, 6, 9, 3, 2, 8, 7, 1]

对于字典的测试：

a {0: 4, 1: 3, 2: 10, 3: 4, 4: 6, 5: 7, 6: 3, 7: 1, 8: 10, 9: 5, 10: 2, 11: 0, 12: 1, 13: 10, 14: 10}

[(0, 4), (1, 3), (2, 10), (4, 6), (5, 7), (7, 1), (9, 5), (10, 2), (11, 0)]

【代码实现】

```python
def deleteDb(t, key=None):
    # 对于字典
    if isinstance(t, dict):      # 判断类型
        s = set()                # 建立集合
        for k, v in t.items():   # 获取字典的key-value值
            if v not in s:       # 第一次出现的元素加入集合
                yield k, v       # 利用生成器，生成元素
                s.add(v)
    else:
        s = set()                # 其他都与dict除了方式一致
        for v in t:
            value = v if key is None else key(v)    # 增加对于不可哈希的序列，进行转换。
            if v not in s:
                yield v
                s.add(value)
```

```python
def show():
    # 列表测试
    d = [random.randint(0, 10) for x in range(20)]
    print('d', d)
    print(list(deleteDb(d)))

    # 字典测试
    a = {i: random.randint(0, 10) for i in range(15)}
    print('a', a)
    new = list(deleteDb(a))
    print(new)
```

题目5　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。

【实现原理】

在了解类的定义和类的一些内置方法的重载方式之后，相对较为容易的构造自己的 Time 类函数。由于时间是有限制的，因此对于构造函数的设置，需要增加一些判断条件。对于一些运算符的重载，需要根据时间的特性，进行进位，格式输出设置等等。

【结果】

a. 构造函数：若输入的时间，不在 24 小时制中，不构造 Time 对象，并显示时间错误。

```
class Time:
    __hour = 0
    __minute = 0
    __second = 0

    def __init__(self, h=0, m=0, s=0):
        try:
            if 0 <= h < 24:
                self.__hour = h
            else:
                raise ValueError
            if 0 <= m < 60:
                self.__minute = m
            else:
                raise ValueError
            if 0 <= s < 60:
                self.__second = s
            else:
                raise ValueError
        except ValueError as e:
            print('%02d:%02d:%02d' % (h, m, s), 'TypeError: Time is not correct.')
```

```
# 测试 构造函数
print('--测试 构造函数--')
t0 = Time(0, 0, 0)
t1 = Time(1, 2, 3)
t2 = Time(4, 5, 6)
t3 = Time(4, 5, 7)
h_error = Time(24, 2, 2)
m_error = Time(12, 67, 2)
s_error = Time(12, 24, 88)
'''
--测试 构造函数--
24:02:02 TypeError: Time is not correct.
12:67:02 TypeError: Time is not correct.
12:24:88 TypeError: Time is not correct.
'''
```

b. 输出重载：利用 format 输出 24 小时制时间

```
# 调用print()函数时，输出__str__返回的内容
def __str__(self):
    return '%02d:%02d:%02d' % (self.__hour, self.__minute, self.__second)
```

```
# 测试 __str__重载:
print('--测试 __str__重载--')
print('t0:', t0)
'''
--测试 __str__重载--
t0: 00:00:00
'''
```

c. 两数相加重载，值接受两个都是 Time 类的变量进行相加，保证时间的正确性，同时设置进位系统。

```
def __add__(self, t):
    if isinstance(t, Time):
        new = Time()
        new.__hour = self.__hour + t.__hour
        new.__minute = self.__minute + t.__minute
        new.__second = self.__second + t.__second

        # 对于两个时间产生的进位进行处理
        while new.__second >= 60:
            new.__second -= 60
            new.__minute += 1
        while new.__minute >= 60:
            new.__minute -= 60
            new.__hour += 1
        while new.__hour >= 24:
            new.__hour -= 24
    else:
        print('TypeError: Only accept Time.')
        raise TypeError
    return new
```

```
# 测试 __add__重载
print('--测试 __add__重载--')
print(t1, '+', t2)
print(t1 + t2)
'''
--测试 __add__重载--
01:02:03 + 04:05:06
05:07:09
'''
```

d. 把时间转换成秒钟

```
    def time2int(self):
        """把事件对象转换为秒数。"""
        return (self.__hour * 60 + self.__minute) * 60 + self.__second
```

```
# 测试 time2int
print('--测试 t1 + t2--')
print('time:', t0, 'second:', t0.time2int())
print('time:', t1, 'second:', t1.time2int())
'''
--测试 t1 + t2--
time: 00:00:00 second: 0
time: 01:02:03 second: 3723
'''
# 测试 printtime
print('--测试 printtime--')
t1.printtime()
'''
```

e. 从小时，分钟，秒钟，逐层判断时间先后并输出结构。

```
def isafter(self, t):
    """ 判断两个事件对象的先后"""
    try:
        if isinstance(t, Time):
            if t.__hour > self.__hour:
                print(t, "is after")
                return t
            elif t.__minute > self.__minute:
                print(t, "is after")
                return t
            elif t.__second > self.__second:
                print(t, "is after")
                return t
            else:
                print(self, "is after")
                return self
        else:
            raise TypeError
    except TypeError as e:
        print('TypeError: Only accept Time.')
```

f. 先把输入的时间转换为 Time 类对象，再通过调用 add 函数，进行两个时间相加。

```
def increment(self, i):
    try:
        if i > 0 or isinstance(i, Time):
            import math
            h = i // 3600
            m = (i - h * 3600) // 60
            s = i - h * 3600 - m * 60
            t = Time(h, m, s)
            return self.__add__(t)
        else:
            raise ValueError
    except ValueError as e:
        print('ValueError: seconds less than zero.')
```

g. 判断时间的有效性，类似构造函数中的操作。

```
def isvalid(self):
    if 0 <= self.__hour < 24 and 0 <= self.__minute < 60 and 0 <= self.__second < 60:
        print('Time is legal.')
    else:
        print('Time is illegal.')
```

【__str__和__repr__分析】

- __Str__是面向用户的打印方式：
  - 只支持使用 print()输出指定格式
  - 在使用 print 输出内容的时候，如何类中同时构造了__str__和__repr__方法，则会优先调用__str__函数。
- __repr__是面对程序员的打印方式：
  - 支持在交互界面不使用 print()输出指定格式
  - 支持使用 print()输出指定格式
- 样例代码

```
>>> class Time:
    __hour = 0
    __minute = 0
    __second = 0

    def __init__(self, h=0, m=0, s=0):
        try:
            if 0 <= h < 24:
                self.__hour = h
            else:
                raise ValueError
            if 0 <= m < 60:
                self.__minute = m
            else:
                raise ValueError
            if 0 <= s < 60:
                self.__second = s
            else:
                raise ValueError
        except ValueError as e:
            print('%02d:%02d:%02d' % (h, m, s), 'TypeError: Time is not correct.')

    # 调用print()函数时，输出__str__返回的内容
    # def __str__(self):
    #     return '%02d:%02d:%02d' % (self.__hour, self.__minute, self.__second)

    def __repr__(self):
        return '%02d:%02d:%02d' % (self.__hour, self.__minute, self.__second)

>>> t = Time(10,23,12)
>>> t
10:23:12
>>> print(t)
10:23:12
```

```
>>> class Time1:
    __hour = 0
    __minute = 0
    __second = 0

    def __init__(self, h=0, m=0, s=0):
        try:
            if 0 <= h < 24:
                self.__hour = h
            else:
                raise ValueError
            if 0 <= m < 60:
                self.__minute = m
            else:
                raise ValueError
            if 0 <= s < 60:
                self.__second = s
            else:
                raise ValueError
        except ValueError as e:
            print('%02d:%02d:%02d' % (h, m, s), 'TypeError: Time is not correct.')

    # 调用print()函数时，输出__str__返回的内容
    # def __str__(self):
    #     return '%02d:%02d:%02d' % (self.__hour, self.__minute, self.__second)

    def __str__(self):
        return '%02d:%02d:%02d' % (self.__hour, self.__minute, self.__second)

>>> t1 = Time1(10,23,12)
>>> t1
```

可以观测到代码输出的不同。

题目6 马尔可夫文本分析和应用

a. 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。

b. 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。

c. 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。

【实现原理】

将 n 阶的单词作为字典的 key，n 阶单词后一个出现的单词和它出现的次数作为 value 值。对整个文本做一个总体的统计。之后再利用随机输出的方式，从字典中取值，实现生成随机文本。

【结果】

```
I have taken place with nothing about a week at .
Least idea of the maid ; and not know what .
Have endured much credit for safety , and give me .
. Knightley , face expressed herself ; but flattering himself .
```

【代码实现】

- wordListSum():统计一个单词中所有单词出现的总和，用于随机取数

```
def wordListSum(wordList):
    # 求取某个单词中所有后缀出现次数总和，用于随机选择下一个出现的单词
    sum1 = 0
    for word, value in wordList.items():
        sum1 += value
    return sum1
```

- 通过 randint 随机生成一个在总的统计范围内的一个数值，通过遍历字典+randIndex 的变化选择输出的单词。

```
def retrieveRandomWord(wordList):
    # 用于选择生成下一个出现的单词
    randIndex = randint(1, wordListSum(wordList))
    for word, value in wordList.items():
        randIndex -= value
        if randIndex <= 0:
            return word
```

- 对于文字的输入要进行一些必要的处理。使得所建立字典中的文字都是我们希望的到的样子。尤其是对于一些标点符号前后的空格进行了处理。

```
def buildWordDict(text):
    # 剔除换行符和引号
    text = text.replace("\n", " ")
    text = text.replace("\"", "")
    text = text.replace("--", "")
    # 保证每个标点符号都和前面的单词在一起
    # 这样不会被剔除，保留在马尔科夫链中
    punctuation = [',', '.', ';', ':']
    for symbol in punctuation:
        text = text.replace(symbol, " " + symbol + " ")
    words = text.split(" ")
    # 过滤空单词
    words = [word for word in words if word != ""]

    wordDict = {}
    for i in range(1, len(words)):
        if words[i - 1] not in wordDict:
            # 为单词新建一个字典
            wordDict[words[i - 1]] = {}
        if words[i] not in wordDict[words[i - 1]]:
            wordDict[words[i - 1]][words[i]] = 0
        wordDict[words[i - 1]][words[i]] = wordDict[words[i - 1]][words[i]] + 1
    return wordDict
```

- 打开文本，构造字典。

- Length,m 设置了总的长度以及句子个数。
- 通过 str 的方法和循环遍历，设置句子的长度以及首字母大写的效果。

```
# text = str('Today is a good day! I love summer!! Today is a bad day.')
text = open('emma.txt', 'r').read()
wordDict = buildWordDict(text)
# 生成链长为length的马尔科夫链
length = 40
# 句子个数为m句
m = 4
chain = ""
currentWord = "I"
for i in range(0, m):
    chain = ''
    for j in range(0, length // m):
        chain += str(currentWord + ' ')
        # 将每句开头设置为大写
        if j == 0:
            chain = chain.replace(chain[0], chain[0].upper())
        currentWord = retrieveRandomWord(wordDict[currentWord])
    chain += str('.')
    print(chain)
```

题目7 模拟快餐订餐场景

a. 定义 4 个类：Customer 顾客类，Employee 商户类，Food 食物类 以及 Lunch 订餐管理。

b. 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

c. Lunch 类包含 result 方法，要求 Customer 打印所收到的食物订单。

d. 编写交互式界面验证所设计的订餐系统。

【实现原理】

利用 Python 面向对象编程中的，类，类的继承，私有方法，等等内容，实现一个订餐系统。

【结果】

正常订餐情况：+ 输入用户名字

+ 返回等待消息
+（根据时间段）输出问候消息。
+ 输出经理名字
+ 输入 yes,表示要点餐
+ 输入要点的菜品的名字，返回点菜成功消息。

```
Welcome to our restaurant.
Please enter your name:Cetacean
Generating an order,please wait...
Good Evening: Cetacean
We have arranged Mr Service for you.
Do you want to order a dish?(yes/no)yes
Please enter the dish: (name)French fries
Order French fries successfully!!
Do you want to order a dish?(yes/no)yes
Please enter the dish: (name)pizza
Order pizza successfully!!
Do you want to order a dish?(yes/no)no
Do you want to see your orderlist?(yes/no)ye
['French fries', 'pizza']
```

+ 输入 yes 表示要打印菜单，返回菜单列表

非正常订餐情况：+ 输入用户名字

+ 返回等待消息

+（根据时间段）输出问候消息。

+ 输出经理名字

+ 输入 no, 表示不要要点餐

+ 输入 yes 表示要打印菜单，由于没有点任何菜品，所以输出“You haven’t order a dish”的提示。

```
Welcome to our restaurant.
Please enter your name:Lily
Generating an order,please wait...
Good Evening: Lily
We have arranged Mr Service for you.
Do you want to order a dish?(yes/no)no
Do you want to see your orderlist?(yes/no)yes
You haven't order a dish.
```

【代码实现】

a. Lunch 对象初始化：对传入的 Customer, Employee 对象进行类型判断，对于不是该类型的数据抛掷异常，进行异常处理。

```
def __init__(self, c, e):  # lunch init: 传入Customer和Employee实例
    try:
        if isinstance(c, Customer):  # 判断Customer的类型，并赋初值
            self.__cus = c
        else:
            raise TypeError
        if isinstance(e, Employee):  # 判断Employee的类型，并赋初值
            self.__emp = e
        else:
            raise TypeError
    except TypeError as e:
        print('TypeError: Only receive Customer and Employee.')
```

b. Lunch 的功能函数

- order ()：调用 customer 对象的 placeOrder 函数
- result()：输出 Customer 点餐的列表，通过长度判断，对于没有点餐的顾客输出没有点餐的提示。
- __str__()：重载了一个 print()格式，输出用户对于的 Employee 的名字。

```
def order(self, food):
    # Lunch的order()函数，传入参数food(食物名),调用实例customer的对象placeOrder()
    # 给placeOrder()传入参数：Employee实例，和food.
    # print('enter LunchOrder.')
    return self.__cus.placeOrder(self.__emp, food)

def result(self, c):
    # 输出用户所定的菜品，没有订单时输出没有订餐提示。
    if isinstance(c, Customer):
        print(c.getlist()) if len(c.getlist()) > 0 else print('You haven\'t order a dish.')

def __str__(self):
    # 输出安排Employee实例的名字
    return 'We have arranged ' + self.__emp.getName() + ' for you.'
```

c. Customer 类的功能函数：

- placeOrder():对传入参数判断类型，并且将下单的菜品存入用户对应的菜单数组中。
- getlist():设置一个 get 方法获取订单数组。

```
# 调取Employee，进行下单，对于下单成功的菜品加入orderlist[]列表。
def placeOrder(self, e, food):
    # print('enter placeorder.')
    if isinstance(e, Employee):
        self.__orderlist.append(e.takeOrder(food).getName())
        return e.takeOrder(food)

def getlist(self):
    return self.__orderlist
```

引入 time 库，根据用户登录点菜系统的时间，输出对应不同时间段的问候语句。

```
# 输出问候句，引入time库，根据时间输出不同的问候
def __str__(self):
    import time
    if 0 < time.localtime().tm_hour < 12:
        return 'Good Morning: ' + self.__name
    elif 12 <= time.localtime().tm_hour < 18:
        return 'Good Afternoon: ' + self.__name
    elif 18 <= time.localtime().tm_hour <= 24:
        return 'Good Evening: ' + self.__name
```

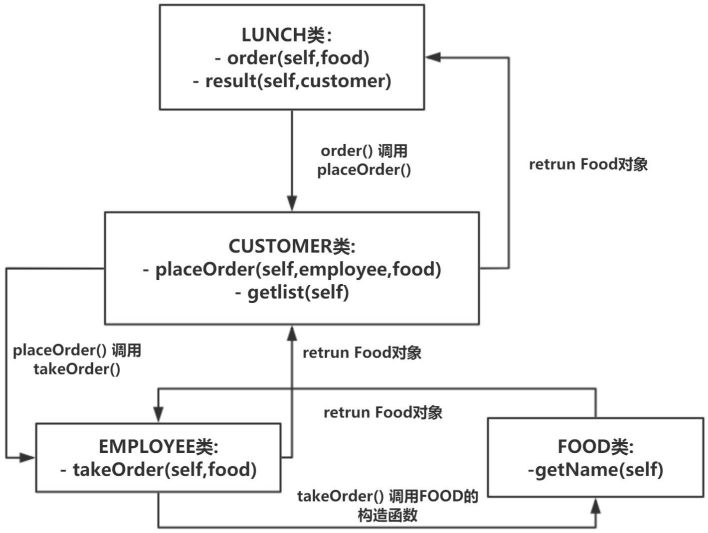
d. Employee 的实现

```
class Employee:
    def __init__(self, name='Mr Service'):
        self.__name = name

    # 返回一个Food类
    def takeOrder(self, food):
        # print('enter takeorder.')
        return Food(food)

    def getName(self):
        return self.__name
```

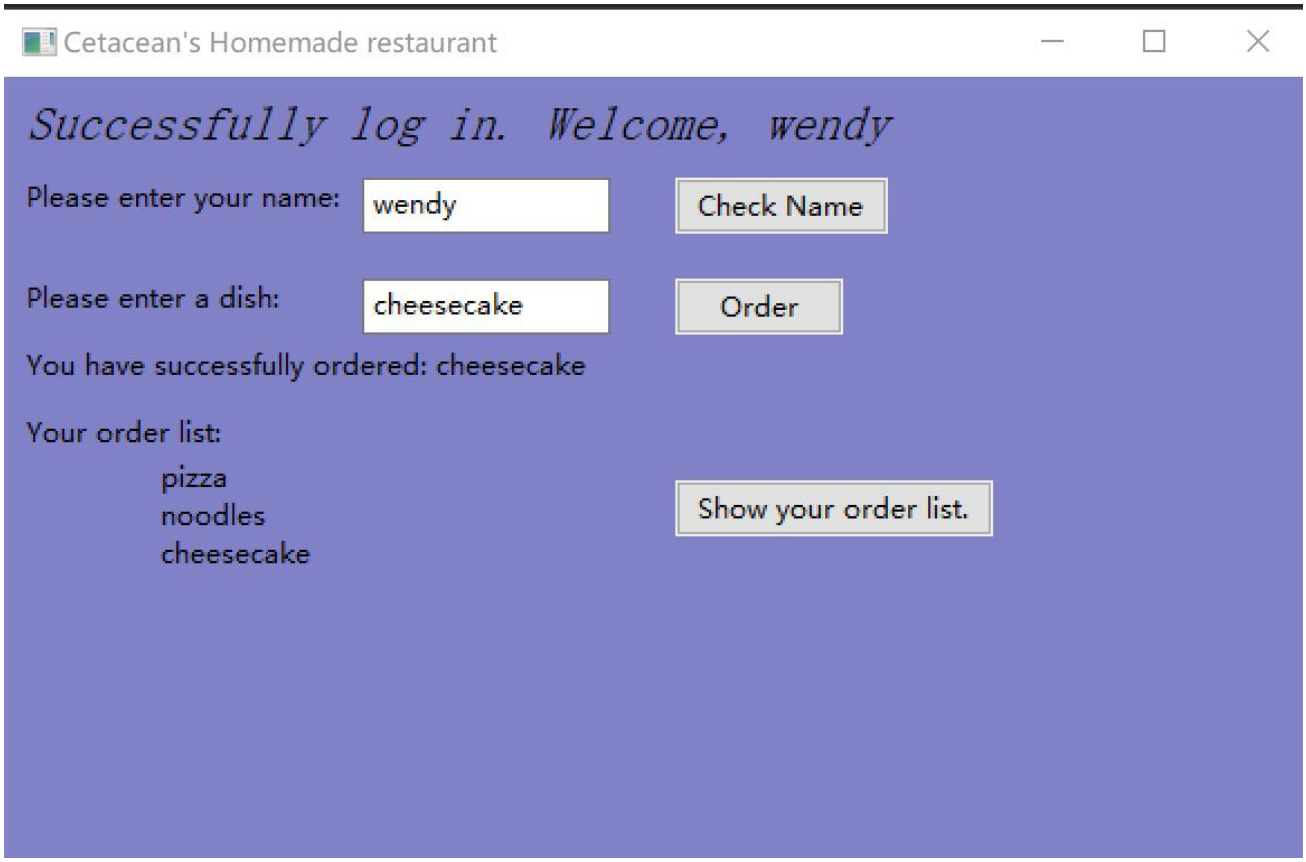
e. Food 的实现

```
class Food:
    def __init__(self, food):
        self.__food = food

    # 输出下单成功消息
    def __str__(self):
        return 'Order ' + self.__food + ' successfully!!'

    def getName(self):
        return self.__food
```

【类的调用流程图】



【GUI 编写可视化窗口】



五、 测试用例

Emma.txt 和 whitefang.txt

六、 收获与体会

次实验涵盖的内容非常丰富，不仅温故了之前的内容，也利用以及学习到的 Python 编程知识完成了对于文本语言缝隙的内容。此外，还巩固了新学的关于面向对象编程的内容，帮助我更好的了解类的私有成员，不同类中调用，初始化等等之间的关系和影响。

第一个实验看似知识一个简单的绘图的问题，不论如何编写程序，繁琐或简单最终其实都能完成题目所要求的绘制。但是其实这道题让我学习到的是一种编程思想，将实际问题转换为程序化和模式化的步骤，同时也需要化整为零，把复杂的东西，化简成我们已知的内容。然后，结合计算机编程的特点将其完成。

关于实验中的两道有关于类的问题，帮助我更好的了解了面向对象各种不同成员所能访问操作的权限，以及有关于类的初始化的问题。在设计订餐类的过程中，最开始对于不同类之间的调用，发送信息的过程一度是比较迷茫的。再反复阅读题目，了解了题目之间的关系之后，才逐渐了解到本质是不同对象方法的调用，以及对于对应类成员函数的修改和显示。明白它的原理之后，编写程序的过程就变得顺利很多。还对于一些类，增加了提示语和时间问候语，感觉利用已经学习到的知识优化完善代码也是一件很有乐趣的事情。

对于剩余的几道实验题，有从互联网中寻找阅读了一些相关资料。了解到它们的算法原理本身是什么，然后再思考如何利用代码实现算法思想。这个过程给我更多的帮助是当我面对一个完全陌生的概念的时

候，该如何借助网络的帮助学习新的知识，进而调用已有的知识完成实验内容。增强了我的自学能力。

综上，本次实验我的收获颇丰，学习到了很多和 Python 相关更偏向于应用类型的编程方式。