上海大学
SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号	第 10 组
实验序号	实验三
学　　号	19122206
姓　　名	陈宝润
日　　期	2021 年 5 月 22 日

一、 实验目的与要求

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、 实验环境

1. 操作系统：windows10

2. Python IDLE、VS Code

三、 实验内容

（一） 验证实验：

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：

1. （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。

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

1. （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。

四、 验证实验的内容分析

1. 实验 6

（1） 代码更改：

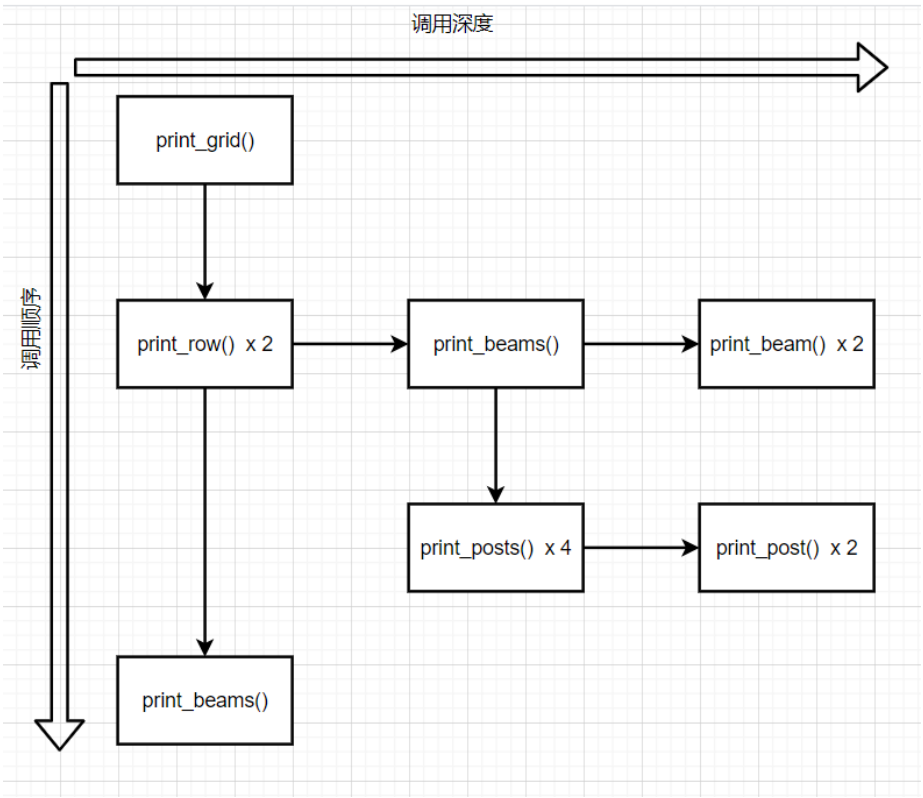
对于 Grid.py，我们只需修改 print()函数即可是该文件支持 python3，对 print 加上括号，如果某处利用，控制 print 输出不换行，更改时，须在函数中加上参数 end = ‘’。

（2） 流程图：

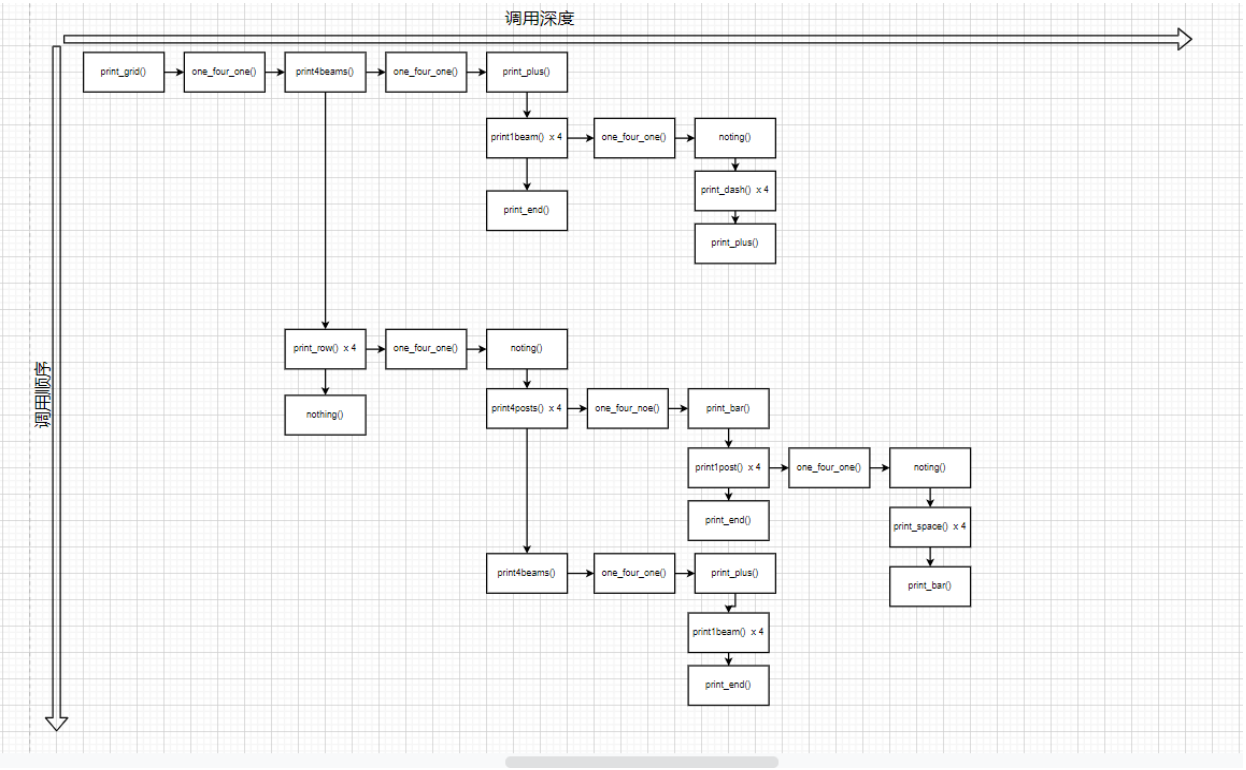
对于 two-by-two version of grid，流程图如图 1：

对于 four-by-four grid，流程图如图 2

图形原件打包至压缩包中。



图表 1流程图 1



图表 2流程图 2

（3） 学习体会：

在绘制 4x4 表格之时，作者清楚的分析了绘制流程，总结规律，然后讲相同的操作封装为函数，代码重用，极大减少代码量与阅读难度。

2. 实验 7

7 中两份代码 myArray.py 和 myMatrix.py 分别实现了一维向量和矩阵的存储。将各种操作封装在类中，数据都用列表存储，但是在 myMatrix.py 文件中的 simNumpyArray 类中，添加了两个属性，行数与列数。

然后，类中重载了很多特殊方法，比如各种运算符（+，*，%， in 等等）满足各种运算，又比如__repr__ 和__str__方法，提供更友好的用户交互界面。

3. 实验 8

（1） bug 及其原理分析：

我们在创建了两个 Kangaroo 对象后，对其中一个对象添加内容，对另一个对象不做任何操作，然而，当输出两个对象时，输出的内容确实一样的，为什么呢？

我们可以看到，在类中的__init__(self, contents=[])中，使用了默认参数，默认值为一个空列表，同时，构造的实例的成员变量 pouch_content 指向这个列表，然而，默认参数只创建一次，加之列表为一个可变序列，所以，两个实例的成员变量均指向同一列表。

（2） 修改 bug：

现在，我们有两种方式，第一种方式就是让参数的默认值指向一个不可变的对象，比如 None：

```
def __init__(self, contents=None):
    if contents is None:
        contents = []
    self.pouch_contents=contents
```

又或者是构造一个新的列表，使多个对象无法指向同一列表:

```
def __init__(self, contents=[]):
    self.pouch_contents = [i for i in contents]
```

但是，这种复制知识浅复制而已，如果一个对象改变列表中的内容（可变序列），那么，其它对象的内容也可能会被篡改，所以，这种方式不太可取。

当然，若执意采用第二种方式，需导入 copy 库来进行深复制。

五、 实验内容的设计与实现

1. 实验内容 1

（1） 公式分析即程序架构

对于（1），根据表达式直接写出递归即可，递归函数的一大好处就是代码清晰明确，可读性高。

对于（2），移除序列重复项，由于集合无重复项，元组为不可变序列，所以，只需考虑对列表和字典进行操作即可。对于列表，我们可以利用集合无重复项的特点来判断重复项，即每遍历一个元素，将其存储在一个集合中，判断后面的元素，是否在集合中出现，若是，则说明重复，删除，若不是，则将其存入集合，遍历下一个元素。同时，我们可以用生成器 yield 来完成这一操作。对于字典，由于字典无顺序，所以无需在意顺序是否改变。

（2） 程序初次架构

```
def operlist(listobj):
    #记录已经出现的元素
    tempset = set()
    for x in listobj:
        if x not in tempset:
            tempset.add(x)
            yield x
```

```
tempset = set()
    klist = list(dictobj.keys())
        if dictobj[k] in tempset:
            del dictobj[k]
        else:
            tempset.add(dictobj[k])
```

在左图中，我们用 yield 语句将 x 返回，在此处，yield 语句没有明显的优点，在此只为练习 yield 的使用。右图中，由于在对字典的操作过程中，字典一直在变动，所以，我们需先用一个列表存储字典的键，然后通过遍历这个列表来遍历字典，若遇重复，删除即可。

2. 实验内容 2

（1） 分析实验内容

这个小实验比较常规，就是编写一个 Time 类，存储一个时间，然后编写函数，给类补充方法。首先，类的私有属性应该包括时(__hour)、分(__minute)、秒(__second)。然后编写方法完善类，比如 timetoint()方法，将时间对象转换成秒数，当然，同样也可以编写 inttotime()方法，将秒数转换成时间。然后，为了更良好的用户界面，需要重写特殊方法：__str__()。完成类的功能，重载__add__()方法。进一步完善该类，用户通常看见的时间格式为，23:13:45 此种格式，所以，如果用户的输入为此种格式，我们需要编写方法，通过这种方法，以这种格式的字符串创建实例。

（2） 编写程序

编写类方法相对而言，就没有什么难度了，在之前的课程中，我们都曾编写过函数，实现这些功能，此处只是将这些函数整理、放入类中。

我们需要合理使用@classmethon 和@staticmethod 等装饰器来“装饰”方法。编写方

法 construct_string()，利用用户输入的字符串“xx-xx-xx”构造 Time 实例，宜使用@classmethon 装饰，因为，这是在构造实例之前调用。

```
@classmethod
def construct_string(cls, string_data):
    hour, minute, second = string_data.split(':')
    if cls.isvalid(hour, minute, second):
        time = cls(hour, minute, second)
        return time
    else:
        print("时间不合法，构造失败")
        return Time(0, 0, 0)
```

判断时间是否合法的方法 invalid()宜使用@staticmethod 来修饰，因为我们只需判断 Time 的时分秒是否合法，使用@staticmethon 装饰，使用起来更方便、更灵活。

重载__str__()方法，返回一个字符串，打印实例的信息，所以，我们尽量打印出 xx:xx:xx 此种格式的字符串

3. 实验内容 3

（1） 设计流程，确定程序结构

马尔可夫模型，可以根据当前的状态，推测将来的状态。在次实验中，进行马尔可夫文本分析，就是根据当前的单词，推测下一个单词，由此生成文章。N 阶马尔可夫分本分析，就是指过程中某一时刻的状态只依赖于其前 n 个状态。

可以利用类来封装马尔可夫模型及其相关的处理函数。

首先，我们用一个数据结构来存储文本中出现的单词，由于集合不含重复项，且没有顺序，显然无法用来存储单词，字典是键值对的组合，在此，也不合适，列表和元组满足要求，在此，我选择了列表来存储单词，因为列表可变，便于操作。同时，考虑到对于不同的 Markov 实例对象，所用的文本为同一个，所以，单词列表应为类的公共属性。

然后，对于一个 Markov 对象，需要根据阶数，生成当前状态与下一状态的关系，这里的状态指当前输出的单词。所以，这个关系应为实例属性，每个对象“个人“享有，我们用字典来存储这种关系，键为当前的状态，对应的值也为字典，该”子字典“的键为下一状态的所有可能情况，值为出现这种状态的概率。

然后，生成马尔可夫链，就是给定一个初始状态，然后利用前面所生成的关系，生成下一状态。

（2） 编写程序

首先打开并读取文件，存储为字符串格式，同时，为了方便，我们可以删去该字符串中的空白字符，以及‘[’, ‘]’, ‘*’, ‘#’等无效的字符。

```
text = re.sub(r'[\s *\[\]#-]+', ' ', str(text))
```

然后，处理标点，我们让标点符号紧接着前一个单词。

```
punctuation = [',', '.', ';',':','?', '!']
for symbol in punctuation:
    text = text.replace(symbol, ""+symbol+" ")
```

然后将这个字符串以空白符为分割字符分开，存储在列表中，该列表为 Markov 类的属性。

而后，创建一个 Markov 实例，该实例需要给出参数 n 值，表明这个实例为 n 阶马尔可夫文本分析模型。

在一个实例中，存储着表示当前状态和下一状态关系的字典，由上一节可知，键为当前的状态，对应的值也为字典，该”子字典“的键为下一状态的所有可能情况，值为出现这种状态的概率。n 阶马尔可夫文本，键应为 n 个单词，值得注意的是，字典的键值应为不可变序列，所以，键不可为列表。在此，我将 n 个单词组成的字符串当作键。

```
key = ''
for word in Markov.__wordslist[start:start+self.__n]:
        key += word + ' '
```

然后，读取一个键，然后 start+1，更新键，即为遍历列表，找到所有键。然后，将键的下一个单词存在值所在的字典中，实现较为简单。

在生成马尔可夫链中，需要随机生成第一个键，然后基于这一个键，找到下一个单词，然后，更新键。由于第一个键最好为一个句子的开头，我们可以这样选择：

```
生成随机数 n

l = 0

while True:

    判断单词列表中第 n+i 个单词是否以句号、问号、感叹号结尾：

    如果是，break

    如果不是，l += 1

取第 n+i 之后 5 个单词为第一个键
```

依照第一个键，即为当前状态，生成下一状态，再更新状态。难点在于如何生成下一状态。在关系字典中，当前状态对应的下一状态我们用了字典存储，该子字典的键为下一状态，值为下一状态出现的次数。在此，我们用以下方式处理：

```
求出当前状态的下一状态出现的总数 sum
while True:
    生成一个随机数 n，最小为 1，最大为 sum
    遍历下一状态子字典，如果值大于 n:
        选取当前的值对应的键位下一状态，返回
```

找到下一状态后，更新键，重复生成步骤。我们可以根据生成句子的总数为标准，适时结束生成。判断是否已经生成一个句子，只需判断当前“句子”的最后一个单词的最后一个字符是否为句号、问号等。

4. 实验内容 4

（1） 实验分析与程序逻辑架构

首先，我们需要定义 4 个类，Customer，Employee，Food，Lunch，然后对各类添加方法，完善类，模拟下单。

一个 Lunch 实例中，应含有 Customer 与 Employee 实例，具有下单方法 Order。Employee 类中，应包含 ShowFood、takeOrder、ShowOrder 方法，给顾客显示菜单，以及显示已选的菜品，还有给顾客提供下单方法。Customer 类中，应有 myfood 属性，表示自己的菜单。Food 类应包含类属性 fooddict，表示该快餐店的菜单，同时需要有 ShowAllfood 方法，显示菜单，该方法为类方法，当然，还需有 ShowSelected 方法，显示已选的菜品。

（2） 程序逻辑

Lunch 类中，在__init__()方法中，添加两个属性，分别为顾客(thisCustomer)与服务员(thisEmployee)。同时，在 order 方法中，调用 thisCustomer 对象的 placeOrder 方法，传

递的参数为 thisEmplyee。

在 Customer 类中，又 placeOrder 方法，接收 Lunch 实例传来的 thisEmployee 实例。

```
def placeOrder(self, myEmployee):
    myEmployee.ShowFood()
    self.myfood = myEmployee.takeOrder()
    myEmployee.ShowOrder(self.myfood)
```

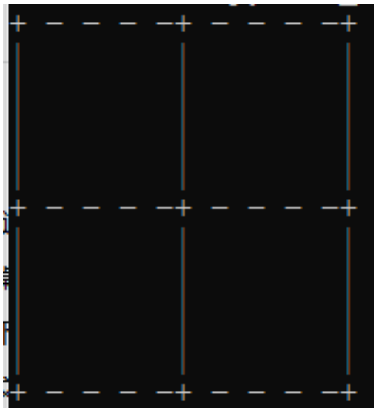
可以看见, myEmployee 对象会调用方法 takeOrder, 来进行点餐, 同时返回一个 Food 对象，将其作为 thisCustomer 的实例属性。而在 Employee 类中，三个方法，ShowFood，takeOrder，ShowOrder 只是调用 Food 类中的方法，分别显示 Food 的类属性，构造和修改一个实例，显示一个实例 Food 的实例属性。

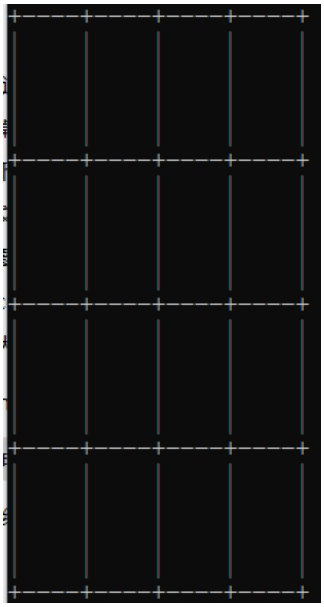
在 Food 类中，菜单用字典来存储，字典的键为菜品所对应的编号，值为菜品。由于 Food 中的方法 ShowAllfood 不需要创建实例，所以把着方法用“@classmethon”装饰。

六、 测试用例

1. 验证实验 6

更改后，运行代码，即可得到两个图，分别为 2by2 和 4by4 表格





2. 验证实验 8

修改 bug 以前，我们可以看到源代码中的 kanga 和 roo 的输出内容相同，更改 buug 后，再次运行代码，可以看到，roo 和 kanga 的输出完全不同：

```
roo:
<__main__.Kangaroo object at 0x0000026228298F10> with pouch contents:


kanga:
<__main__.Kangaroo object at 0x0000026228298F70> with pouch contents:
    'wallet'
    'car keys'
```

3. 设计实验 1

生成字典与列表，对函数进行测试，可得到以下输出：

```
字典测试:
原字典: {0: 2, 1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 2}
更改后: {0: 2}


列表测试:
原列表: [2, 3, 2, 2, 2, 2, 3, 1, 3, 2]
更改后: [2, 3, 1]
```

可以看出，函数运行正常，功能符合实验要求。

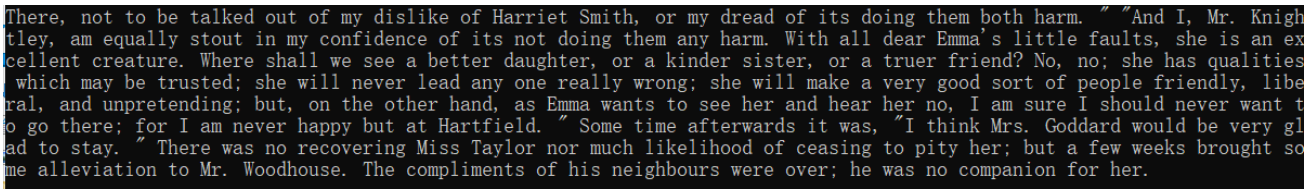
4. 设计实验 2

设计测试用例，然后，我们可以得到以下输出结果：

```
time1: 12:13:12
time2: 9:24:50


Test time1.timetoint()
43992


Test time1.increment(4)
12:13:16


Test time1+time2 and __str__
21时38分2秒
```

5. 设计实验 3

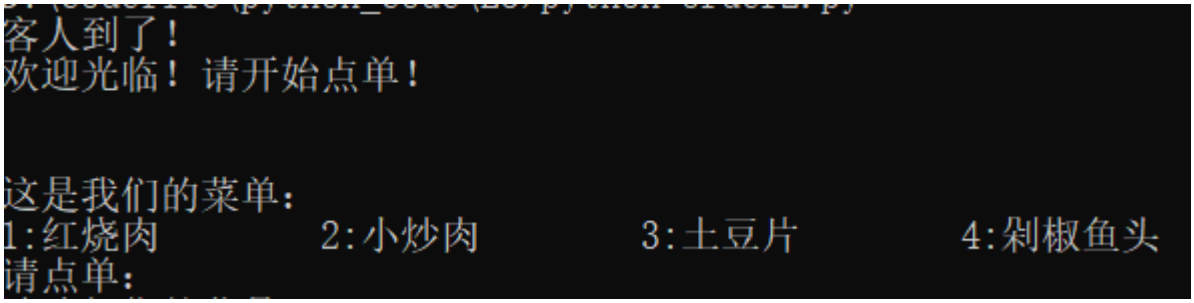
我们利用 emma.txt 训练 Markov 模型，我们可以构建一个 3 阶马尔可夫模型实例，输出 10 个句子，我们可以得到以下输出：



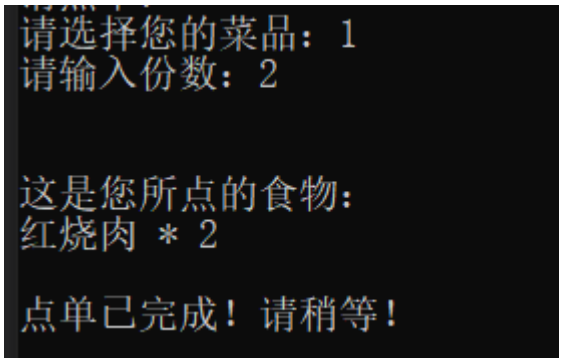
由于只是单纯的字符串输出，所以，会出现一个单词未输出完全便换行的情况，问题有待解决。

6. 设计实验 4

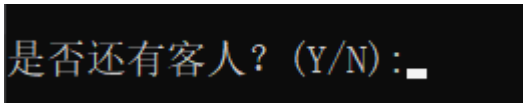
运行程序，会显示菜单的基本信息



随后，程序提醒你输入菜品编号即份数，输入成功后：



这一订单完成，随后，程序询问，是否还有客人：



若输入 y，则继续一次点单流程，若输入 n，则程序结束。

七、 收获与体会

第三次实验，重点应该在于面向对象的编程。培养面向对象编程的意识。

在第三次实验中，我熟悉了类的书写，学会合理使用不同的装饰器。同时，合理利用类属性、实例属性。

但是，自己在类的继承、多态两方面的了解与练习还有所欠缺。