Problem Set 4 作业报告

17341097 廖永滨

一、实验内容

1. 用 python 编写一个获取给定字符串全排列的方法
2. 用 python 编写几个能给字符串加密或解密的类以及对应成员、方法
3. 在 1 和 2 的基础上，更改替换规则（元音字母乱序替换），编写能对给定的字符串进行解密的类以及成员、方法

二、实验目的

根据文档提示，按步骤完成三个部分的程序，PS4A 学习递归函数，PS4B 和 PS4C 学习类与继承的使用。

三、实验要求

1.Part A 要按指定的排列顺序进行排列，且必须用递归的方法。
2.Part B 和 Part C 根据类的各个函数的用法要求，完善类中的各个函数

四、实验流程

Part A：

```
def get_permutations(sequence):
    if len(sequence) <= 1:
        return [sequence]
    sl = []
    for i in range(len(sequence)):
        for j in get_permutations(sequence[0:i] + sequence[i + 1:]):
            sl.append(sequence[i] + j)
    return sl
```

利用递归算法，完成示例程序中的 get_permutations()函数。

1. 当字符串只有一个字母时直接返回
2. 但字符串不只一个时，依次决定第一个字母的位置，对余下的字母进行相同的排列操作从而形成递归

```
example_input = 'ok'
print('Input:', example_input)
print('Expected Output:', ['ok', 'ko'])
print('Actual Output:', get_permutations(example_input))

example_input = 'orz'
print('Input:', example_input)
print('Expected Output:', ['orz', 'ozr', 'roz', 'rzo', 'zor', 'zro'])
print('Actual Output:', get_permutations(example_input))

example_input = 'nice'
print('Input:', example_input)
print('Expected Output:',
      ['nice', 'niec', 'ncie', 'ncei', 'neic', 'neci', 'ince', 'inec', 'icne', 'icen', 'ienc', 'iecn', 'cnie', 'cnei',
       'cine', 'cien', 'ceni', 'cein', 'enic', 'enci', 'einc', 'eicn', 'ecni', 'ecin'])
print('Actual Output:', get_permutations(example_input))
```

测试样例如上，分别是 ok、orz、nice 三个词，结果均符合期望。

Part B: Cipher Like Caesar

这一部分，题目要求完成关于凯撒密码的加密以及解密的类相关的所有方法并通过测试

Message 类：

基础的 Message 类，有基本的字符串属性和对应的 get 方法。

1. __init__(self, text)，初始化 Message 对象 ：
   self.message_text 是一个字符串，
   self.valid_words 是一个列表(load_words 导入文件中的单词)。

```
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words(WORDLIST_FILENAME)
```

2. Get 相关方法 ：
   这里在 get 列表（字典）时，需要进行深拷贝复制源列表（字典）被修改。

```
def get_message_text(self):
    #由于strings对象不可变，可以不拷贝直接返回
    return self.message_text

def get_valid_words(self):
    #深拷贝副本保护源列表
    return copy.deepcopy(self.valid_words)
```

3. build_shift_dict(self, shift)，创建一个字典偏移映射表。
   此处利用 dict 以及 zip 方法可以快速创建字典，zip 方法自动将左边的参数映射到右边对应位置的参数，非常方便。而 dict 方法是 python3 创建字典的一种方法。Update 函数将可以将参数更新进字典里。

```
def build_shift_dict(self, shift):
    dict_low = dict(zip(string.ascii_lowercase, string.ascii_lowercase[shift:] + string.ascii_lowercase[:shift]))
    dict_high = dict(zip(string.ascii_uppercase, string.ascii_uppercase[shift:] + string.ascii_uppercase[:shift]))
    dict_high.update(dict_low)  #合并
    return dict_high
```

4. apply_shift(self, shift)，将密码的映射规则应用于对象的 message_text 属性，返回加密后的信息。要忽略空格和标点符号。

利用 in 语法判断是否在特殊符号里，是则直接拼接，否则拼接映射后的结果。

```
def apply_shift(self, shift):
    # 利用self.build_shift_dict方法得到映射字典
    dict_map = self.build_shift_dict(shift)
    # 题目要求忽略的标点符号
    symbols = " !@#$%^&*()-_+={}[]|\:;'<>?,./\""
    # 根据偏移获取要求的加密字符串，特殊符号不偏移
    result = ''
    for letter in self.get_message_text():
        if letter in symbols:
            result = result + letter    #直接拼接
        else:
            result = result + dict_map[letter]  #偏移后拼接
    return result
```

PlaintextMessage 类：

这是 Message 类的子类，要对给定字符进行加密

1. __init__(self, text)，初始化 PlaintextMessage 对象 ：

根据传入的字符串和偏移量初始化 PlaintextMessage 类属性，要初始化继承来的父类属性，故要调用父类的 init 函数。

```
def __init__(self, text, shift):
    #继承的父类要进行初始化
    Message.__init__(self, text)
    #偏移量
    self.shift = shift
    # 调用继承来的build_shift_dict函数得到映射表
    self.encryption_dict = self.build_shift_dict(shift)
    # 调用继承来apply_shift得到加密后的字符串
    self.message_text_encrypted = self.apply_shift(shift)
```

2. Get 类方法，同上一个类，要保护源对象。

```
def get_shift(self):
    return shift

def get_encryption_dict(self):
    #深拷贝保护源字典
    return copy.deepcopy(self.encryption_dict)

def get_message_text_encrypted(self):
    return self.message_text_encrypted
```

3. change_shift(self, shift)，改变对象属性中的偏移量。
   利用修改后的值重新初始化类属性即可。

```
def change_shift(self, shift):
    #利用修改后的shift重新初始化类对象
    self.__init__(self.get_message_text(), shift)
```

4. change_shift(self, shift)，改变对象属性中的偏移量。

CiphertextMessage 类：

这是 Message 类的子类，将对应字符串进行解密。

1. __init__(self, text)，初始化 CiphertextMessage 对象 ：
   根据传入的字符串和偏移量初始化 CiphertextMessage 类属性，要初始化继承来的父类属性，故要调用父类的 init 函数。

```
def __init__(self, text):
    Message.__init__(self, text)
```

2. decrypt_message(self)，解密函数。
   通过加密一样就可以实现解密，遍历一遍 26 个字母。

```
def decrypt_message(self):
    # 最好的偏移量
    best_shift = 0
    # 最多合法的单词数
    max_words = 0
    # 解密后最好的字符串
    best_msg = ''
    # 遍历0-25个偏移量
    for i in range(26):
        num = 0             #偏移i单位的匹配单词数
        temp = self.apply_shift(i)  #偏移i单位的解密字符串
        '''
        将所有单词都分割出来，此处只处理正常语法的message，也就是词与词之间必有空格
        若要将类似 hello，world 这种词与词之间没有空格而只有标点符号的字符串需要import re，用re里的方法，此处不考虑这种情况
        '''
        temp_list = temp.split()
        # 计算合法单词数
        for word in temp_list:
            if is_word(self.get_valid_words(), word):
                num += 1
        # 如果这个偏移量得到的单词数，比当前最大的还大就改变
        if num > max_words:
            max_words = num
            best_shift = i
            best_msg = temp

        # 如果合法单词数等于全部单词数，直接返回
        if num == len(temp_list):
            break

    return (best_shift, best_msg)
```

测试样例：

```
# Example test case (PlaintextMessage)
    plaintext = PlaintextMessage('I am very happy', 2)
    print('Expected Output: K co xgta jcrra')
    print('Actual Output:', plaintext.get_message_text_encrypted())
    # Example test case (CiphertextMessage)
    ciphertext = CiphertextMessage('K co xgta jcrra')
    print('Expected Output:', (24, 'I am very happy'))
    print('Actual Output:', ciphertext.decrypt_message())
    # Example test case (PlaintextMessage)
    plaintext = PlaintextMessage('I am so sad', 10)
    print('Expected Output: S kw cy ckn')
    print('Actual Output:', plaintext.get_message_text_encrypted())
    # Example test case (CiphertextMessage)
    ciphertext = CiphertextMessage('S kw cy ckn')
    print('Expected Output:', (16, 'I am so sad'))
    print('Actual Output:', ciphertext.decrypt_message())
```

如上，解密与加密测试，输出与期望一致

Part C：Substitution Cipher

这一部分，题目要求完成关于凯撒密码的元音替换加密以及解密的类相关的所有方法并通过测试。

SubMessage 类：

1. __init__(self, text)，初始化 SubMessage 对象 ：
   self.message_text 是一个字符串，
   self.valid_words 是一个列表(load_words 导入文件中的单词)。

```
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words(WORDLIST_FILENAME)
```

2. Get 相关方法 ：
   这里在 get 列表（字典）时，需要进行深拷贝复制源列表（字典）被修改。

```
def get_message_text(self):
    return self.message_text

def get_valid_words(self):
    return copy.deepcopy(self.valid_words)
```

3. build_transpose_dict(self, vowels_permutation)，创建一个字典，根据输入的全排列，返回加密前后的字母的对应关系。

同 PartB，利用 dict 和 zip 方法建立映射字典。

```
def build_transpose_dict(self, vowels_permutation):
    dict_low = dict(zip(CONSONANTS_LOWER, CONSONANTS_LOWER))
    dict_high = dict(zip(CONSONANTS_UPPER, CONSONANTS_UPPER))
    dict_low_vowel = dict(zip(VOWELS_LOWER, vowels_permutation.lower()))
    dict_high_vowel = dict(zip(VOWELS_UPPER, vowels_permutation.upper()))
    dict_high.update(dict_low)  #合并
    dict_high.update(dict_low_vowel)
    dict_high.update(dict_high_vowel)
    return dict_high
```

4. apply_transpose(self, transpose_dict)，将替换密码的规则应用于对象的 message_text 属性，然后得到加密后的信息。要忽略空格和标点符号。

```
def apply_transpose(self, transpose_dict):
    symbols = " !@#$%^&*()-_+={}[]|\:;'<>?,./\""
    result = ''
    for letter in self.get_message_text():
        if letter in symbols:
            result = result + letter
        else:
            result = result + transpose_dict[letter]
    return result
```

EncryptedSubMessage 类：

SubMessage 类的字类，将信息通过替换加密的规则进行解密。

1. __init__ 根据传入的字符串和偏移量初始化 EncryptedSubMessage 类属性

```
def __init__(self, text):
    SubMessage.__init__(self, text)
```

2. decrypt_message(self)，通过遍历元音字母全排列的方式找到最多合法字母的转换方案，也就是题目要求的方案。

```
def decrypt_message(self):
    # 获取映射字典
    perm_list = get_permutations(VOWELS_LOWER)
    # 最大合法单词数
    max_words = 0
    # 当前最好的匹配字符串
    best_msg = self.get_message_text()
    # 遍历每个组合
    for perm in perm_list:
        num = 0
        # 创建映射字典
        temp_transpose_dict = self.build_transpose_dict(perm)
        # 解密 - 反向加密
        temp = self.apply_transpose(temp_transpose_dict)
        # 字符串通过空格分割成单词组,同ps4b, 只分割带空格的
        temp_list = temp.split()
        # 判断合法单词数
        for word in temp_list:
            if is_word(self.get_valid_words(), word):
                num += 1
        # 如果合法单词数比当前多
        if num > max_words:
            max_words = num
            best_msg = temp
        # 如果合法单词数等于单词总数就退出
        if num == len(temp_list):
            break
    return best_msg
```

测试样例 ：

输出能全部通过测试，说明类实现无问题

```
# Example test case
message = SubMessage("aeiou")
permutation = "iouea"
enc_dict = message.build_transpose_dict(permutation)
print("Original message:", message.get_message_text(), "Permutation:", permutation)
print("Expected encryption:", "iouea")
print("Actual encryption:", message.apply_transpose(enc_dict))
enc_message = EncryptedSubMessage(message.apply_transpose(enc_dict))
print("Decrypted message:", enc_message.decrypt_message())

# Example test case
message = SubMessage("Good job!")
permutation = "ieauo"
enc_dict = message.build_transpose_dict(permutation)
print("Original message:", message.get_message_text(), "Permutation:", permutation)
print("Expected encryption:", "Guud jub!")
print("Actual encryption:", message.apply_transpose(enc_dict))
enc_message = EncryptedSubMessage(message.apply_transpose(enc_dict))
print("Decrypted message:", enc_message.decrypt_message())

# Example test case
message = SubMessage("Sad is happy!")
permutation = "ieauo"
enc_dict = message.build_transpose_dict(permutation)
print("Original message:", message.get_message_text(), "Permutation:", permutation)
print("Expected encryption:", "Sid as hippy!")
print("Actual encryption:", message.apply_transpose(enc_dict))
enc_message = EncryptedSubMessage(message.apply_transpose(enc_dict))
print("Decrypted message:", enc_message.decrypt_message())
```

五、实验总结

1、对 Python 一些操作有了更深入的了解，比如 in，for 等等语法

2、深入了解了深拷贝与浅拷贝的区别并应用到了程序中，程序中多处代码用到了深拷贝。

3、项目的结构思想更加深入，对模块化程序设计有了更好的认识

4、对遍历删除元素有了新认识，由于删除元素会导致迭代器终止，所以要用遍历的手法删除元素就要求多次遍历或者保存迭代信息，为了简单，我采取了前者。

5、对类这一概念有了较为系统全面的了解。