**Name: Session:**

**Programming I**

**Lab Exercise 12.8.2021**

**Encryption and Decryption**

**GETTING STARTED**

This problem set has two parts. Part A deals with encryption, a very important concept in computer science. Part B is a set of problems that is designed to decrypt what you have encrypted.

Download and save the files on the server in the Encryption Project folder. This folder includes the following files:

* encryption.py:

Skeleton code you'll **fill in as you work through this problem**.

* words.txt:

A list of English words (55,909 of them)

* bestShift pseudo.txt:

Pseudocode for the bestShift function.

* story.txt:

An encoded story

Load encryption.py into a Python environment without making any modifications to it, in order to ensure that everything is set up correctly. The code that we have given you loads a list of words from a file. If everything is okay, after a small delay, you should see the following printed out:

Loading word list from file...

55909 words loaded.

If you see an IOError instead (e.g., No such file or directory), you should change the value of the WORDLIST\_FILENAME constant (defined near the top of the file) to the complete pathname for the filewords.txt (this will vary based on where you saved the file).

The file encryption.py has a few functions already implemented that you can use while writing up your solution. You can ignore the code between the following comments, though you should read and understand everything else:

## HAIL CAESAR!

Encryption is the process of obscuring information to make it unreadable without special knowledge. For centuries, people have devised schemes to encrypt messages - some better than others - but the advent of the computer and the Internet revolutionized the field. These days, it's hard not to encounter some sort of encryption, whether you are buying something online or logging into a shared computer system. Encryption lets you share information with other trusted people, without fear of disclosure.

A cipher is an algorithm for performing *encryption* (and the reverse, *decryption*). The original information is called *plaintext*. After it is encrypted, it is called *ciphertext*. The ciphertext message contains all the information of the plaintext message, but it is not in a format readable by a human or computer without the proper mechanism to decrypt it; it should resemble random gibberish to those for whom it is not intended.

A cipher usually depends on a piece of auxiliary information, called a *key*. The key is incorporated into the encryption process; the same plaintext encrypted with two different keys should have two different ciphertexts. Without the key, it should be difficult to decrypt the resulting ciphertext into readable plaintext.

This assignment will deal with a well-known (though not very secure) encryption method called the Caesar cipher. In this problem set you will need to devise your own algorithms and will practice using recursion to solve a non-trivial problem.

## CAESAR CIPHER

The basic idea of the Caesar cipher is that you pick an integer for a key, and shift every letter of your message by the key. For example, if your message was "hello" and your key was 2, "h" becomes "j", "e" becomes "g", and so on, because we are shifting every letter two spots to the right.

We will use a variant of the standard Caesar cipher where we will treat upper and lower case letters separately, so upper case letters will always be mapped to upper case letters, and lower case letters will always be mapped to lower case letters. Thus, if "a" maps to "c", "A" will map to "C". Characters such as the space character, commas, periods, exclamation points, etc will *not* be encrypted by this cipher - basically, all the characters within string.punctuation, plus the space (' ') and all numerical characters (0 - 9). Several helpful string functions can be used to accomplish this such as string.isalpha(), string.isspace(), string.isdigit(), and string.isspace().

## WRAPPER FUNCTIONS

Now that you are ready to start coding, you can look carefully at the code skeletons that I have provided for you. Do not be intimidated by the length of the function specifications I provide with the supplied code! Many of these problems rely on **wrapper functions**, an extremely useful coding concept that, when implemented correctly, often requires very little additional code. The idea of wrapper functions here is that the functions visible to a user take as arguments simple inputs, and then supply these arguments - plus other information - to functions visible only within the implementation to perform the computation.

## PROBLEM 1: ENCRYPTION

You'll now write a program to encrypt plaintext into ciphertext using the Caesar cipher.

**Upper and Lower Case Letters**

Be sure that your dictionary includes both lower and upper case letters, but that the shifted character for a lower case letter and its uppercase version are lower and upper case instances of the same letter. What this means is that if the original letter is "a" and its shifted value is "c", the letter "A" should shift to the letter "C".

**Characters to Ignore**

A reminder from the introduction page - Characters such as the space character, commas, periods, exclamation points, etc will *not* be encrypted by this cipher - basically, all the characters withinstring.punctuation, plus the space (' ') and all numerical characters (0 - 9).

**Test Cases**

buildCoder(3)

{'A': 'D', 'C': 'F', 'B': 'E', 'E': 'H', 'D': 'G', 'G': 'J', 'F': 'I', 'I': 'L', 'H': 'K', 'K': 'N', 'J': 'M', 'M': 'P', 'L': 'O', 'O': 'R', 'N': 'Q', 'Q': 'T', 'P': 'S', 'S': 'V', 'R': 'U', 'U': 'X', 'T': 'W', 'W': 'Z', 'V': 'Y', 'Y': 'B', 'X': 'A', 'Z': 'C', 'a': 'd', 'c': 'f', 'b': 'e', 'e': 'h', 'd': 'g', 'g': 'j', 'f': 'i', 'i': 'l', 'h': 'k', 'k': 'n', 'j': 'm', 'm': 'p', 'l': 'o', 'o': 'r', 'n': 'q', 'q': 't', 'p': 's', 's': 'v', 'r': 'u', 'u': 'x', 't': 'w', 'w': 'z', 'v': 'y', 'y': 'b', 'x': 'a', 'z': 'c'}

buildCoder(9)

{'A': 'J', 'C': 'L', 'B': 'K', 'E': 'N', 'D': 'M', 'G': 'P', 'F': 'O', 'I': 'R', 'H': 'Q', 'K': 'T', 'J': 'S', 'M': 'V', 'L': 'U', 'O': 'X', 'N': 'W', 'Q': 'Z', 'P': 'Y', 'S': 'B', 'R': 'A', 'U': 'D', 'T': 'C', 'W': 'F', 'V': 'E', 'Y': 'H', 'X': 'G', 'Z': 'I', 'a': 'j', 'c': 'l', 'b': 'k', 'e': 'n', 'd': 'm', 'g': 'p', 'f': 'o', 'i': 'r', 'h': 'q', 'k': 't', 'j': 's', 'm': 'v', 'l': 'u', 'o': 'x', 'n': 'w', 'q': 'z', 'p': 'y', 's': 'b', 'r': 'a', 'u': 'd', 't': 'c', 'w': 'f', 'v': 'e', 'y': 'h', 'x': 'g', 'z': 'i'}

Here is the code for buildCoder():

def buildCoder(shift):

"""

Returns a dict that can apply a Caesar cipher to a letter.

The cipher is defined by the shift value. Ignores non-letter characters

like punctuation, numbers, and spaces.

shift: 0 <= int< 26

returns: dict

"""

code = {}

for i in range(65, 91):

value = i + shift

if value > 90:

value -= 26

code[chr(i)] = chr(value)

for i in range(97, 123):

value = i + shift

if value > 122:

value -= 26

code[chr(i)] = chr(value)

return code

## PROBLEM 2: ENCRYPTION

Next, define the function applyCoder, which applies a coder to a string of text.

**Test Cases**

>>>applyCoder("Hello, world!", buildCoder(3))

'Khoor, zruog!'

>>>applyCoder("Khoor, zruog!", buildCoder(23))

'Hello, world!'

Here is the code for applyCoder():

def applyCoder(text, coder):

"""

Applies the coder to the text.Returns the encoded text.

text: string

coder: dict with mappings of characters to shifted characters

returns: text after mapping coder chars to original text

"""

eText = ''

for i in range(len(text)):

if text[i].isalpha():

eText += coder[text[i]]

else:

eText += text[i]

return eText

## PROBLEM 3: ENCRYPTION

Once you have written buildCoder and applyCoder, you should be able to use them to encode strings using the applyShift function.

**Test Cases**

>>>applyShift('This is a test.', 8)

'Bpqaqaibmab.'

>>>applyShift('Bpqaqaibmab.', 18)

'This is a test.'

Here is the code for applyShift()

def applyShift(text, shift):

"""

Given a text, returns a new text Caesar shifted by the given shift

offset. Lower case letters should remain lower case, upper case

letters should remain upper case, and all other punctuation should

stay as it is.

text: string to apply the shift to

shift: amount to shift the text (0 <= int< 26)

returns: text after being shifted by specified amount.

"""

coder = buildCoder(shift)

eText = applyCoder(text, coder)

return eText

## PROBLEM 4: DECRYPTION

Your friend is really excited about the program he wrote for Problem 1. He sends you emails, but they're all encrypted with the Caesar cipher!

If you know which shift key he is using, then decrypting her message is an easy task. If the string message is the encrypted message and k is the shift key he is using, then calling applyShift(message, 26-k) returns her original message. Do you see why?

The problem is, you don't know which shift key he is using. The good news is, you know your friend only speaks and writes English words. So if you can write a program to find the decoding that produces the maximum number of real English words, you can probably find the right decoding.

## PSEUDOCODE

Right now, you should take time to write some pseudocode! Think about an algorithm you could use to solve this problem and write the steps down. Then, you can verify your algorithm with the supplied pseudocode in bestShift\_pseudo.txt before coding.

After you've done writing and checking yourpseudocode, implement findBestShift().This function takes a wordList and a sample of encrypted text and attempts to find the shift that encoded the text. A simple indication of whether or not the correct shift has been found is if most of the words obtained after a shift are valid words. Note that this only means that *most* of the words obtained are actual words. It is possible to have a message that can be decoded by two separate shifts into different sets of words. While there are various strategies for deciding between ambiguous decryptions, for this problem we are only looking for a simple solution.

To assist you in solving this problem, I have provided a helper function, isWord(wordList, word). This simply determines if word is a valid word according to the wordList. This function ignores capitalization and punctuation.

**Using string.split**

You may find the function string.split useful for dividing the text up into words.

>>> 'Hello world!'.split('o')

['Hell', ' w', 'rld!']

>>> '6.00x is pretty fun'.split(' ')

['6.00x', 'is', 'pretty', 'fun']

**Test Cases**

>>> s = applyShift('Hello, world!', 8)

>>>s

'Pmttw, ewztl!'

>>>findBestShift(wordList, s)

18

>>>applyShift(s, 18)

'Hello, world!'

Here is the code for findBestShift():

def findBestShift(wordList, text):

"""

Finds a shift key that can decrypt the encoded text.

text: string

returns: 0 <= int< 26

"""

maxWords = 0

bestShift = 0

count = 0

for shift in range(26):

eText = applyShift(text, shift)

eWords = eText.split(' ')

for word in eWords:

newWord = ''

for i in range(len(word)):

if word[i].isalpha():

newWord += word[i]

if isWord(wordList, newWord):

count += 1

if count >maxWords:

maxWords = count

bestShift = shift

count = 0

return bestShift

## PROBLEM 5: DECRYPTION

Now that you have all the pieces to the puzzle, please use them to decode the file story.txt. In the skeleton file, you will see a method  getStoryString() that will return the encrypted version of the story. Fill in the following method and include as a comment at the end of the problem set your decryption of the story.

Here is the code for decryptStory():

def decryptStory():

"""

Using the methods you created in this problem set,

decrypt the story given by the function getStoryString().

Once you decrypt the message, be sure to include as a comment

your decryption of the story.

returns: string - story in plain text

"""

wordList = loadWords()

text = getStoryString()

bs = findBestShift(wordList, text)

plainText = applyShift(text, bs)

return plaintext

Now you just need to call decryptStory() as such

print decryptStory() and you will be able to read about Jack Florey.

Finally, write a function printOriginal() which prints the ciphertext of story.txt. I will leave that function to you….

Print your final version of encryption.py and attach it to this handout.