OCR A-Level Computer Science

The Programming Project:

Cryptographic Ciphers Program

Name: Grace Xiao Fu Edgecombe

Contents

Analysis

Problem Proposed

Year 12 are in the first year of studying order to achieve their A-Level qualifications but knowing that they will be applying to university soon, many tend to participate in extra-curricular events such as competitions and insight days. I have been approached by a group of year 12 students who want me to help them produce an offline cryptographic cipher program in order to help them when entering the National Cipher Challenge, an annual competition run by the University of Southampton School of Mathematics.

The students that know in previous years, the Caesar cipher and Vigenère cipher have been used quite commonly in the challenges, however they would also like to have an encryption and decryption section for the Baconian cipher due to them also appearing in challenges frequently in recent years. These challenges are primarily decrypting text and cracking an encrypted message in order to gain points and progress through the weekly challenges.

The group will be using the program to enter the data they receive in the form of a challenge from the National Cipher Challenge (NCC) website to encrypt or decrypt, sometimes with the key given and sometimes not (cracking) the data given and relay the output back into the NCC website in order to receive points to progress in the competition. The program will only be used for some challenges as there are some challenges that will use different ciphers not available on the program that is being produced.

Since it is a competition there are time constraints and more points awarded for earlier answers so they would need the program to work as efficiently as possible so they can maximize the amount of challenges they complete as they will also be solving some of the challenges by hand and not.

Stakeholders

About the Stakeholders

The stakeholders are a small team of 4 year 12's, ranging from ages 16-17 going by the team name Cybites, who are entering the National Cipher Challenge, which is a cryptography competition and need a program that will encrypt plain text files, decrypt or attempt to crack an encrypted message depending on the options that they have selected.

The members of Team Cybites are Luke Smith, Anna Matter, Aden Groove, Elena Yue.

The team is made up of two maths students and two computer science students, but the maths students are quite new to the idea of cryptography more complex than the Caesar cipher whereas the computer science students are not confident in their coding abilities for ciphers. Part of the problem is that none of the students go to the same school and all the students in the group only meet up after school in a library for an hour or so every day to

discuss the challenge and solve it therefore the program cannot take too long to solve the problem.

"We need a program that will encrypt and decrypt for the following ciphers; Caesar, Baconian, Vigenère*."*

– Luke Smith, Computer Science Student

Luke Smith is a Computer Science student who is very good at recognising what type of encryption has been used on the original text, ranging from just and so this will be handled by him. He has done the challenge before but got stuck after he couldn't figure out how to decrypt one of the later challenges.

"The program needs to be simple to use as I'm not a computer scientist. Buttons are always reassuring to see! Although it must be a program that can operate offline as we may have limited resources depending on what library we use. One of us can access the code from the website in school and save it for later, but needing the internet constantly is a bit of an inconvenience."

– Anna Matter, Mathematics Student

Anna Matter is a Mathematics student who wants to learn more about ciphers as they are heavily involve mathematics, but she is the one who suggested that this problem be made into a program in the first place as she was aware that the Vigenère cipher in particular takes a lot of time to encrypt and decrypt by hand.

"It might seem obvious, but we definitely need to be able to copy and paste text into input boxes we are normally given plain text or cipher text through the website, but also have a means of copying the result out of the program since in my previous experiences programming, I could never copy the output in a GUI out of the GUI."

– Elena Yue, Computer Science Student

Elena Yue is a Computer Science student who has previous experiencing coding, but she is normally programs without the use of the object-oriented paradigm so she is not confident she can code the solution, especially producing an output in a way that other people would understand.

"It would be helpful if there are cracking features for the ciphers as cracking takes ages to do manually, especially as we don't get much time to solve these together so ideally if it can crack the piece of text in under 2 minutes, that would be great!"

– Aden Groove, Mathematics Student

Aden Groove is a Mathematics student and he has had a brief look at cracking the ciphers by hand, but realized that while Caesar Cipher cracking wouldn't take an unreasonable amount of time to identify and crack the key, decrypting it with a long text length is tiresome as well as the Vigenère Cipher needs a lot of data to be processed in order to return the key and he said the processes required a lot of complexity not capable by hand.

Background of National Cipher Competition

Structure of Competition

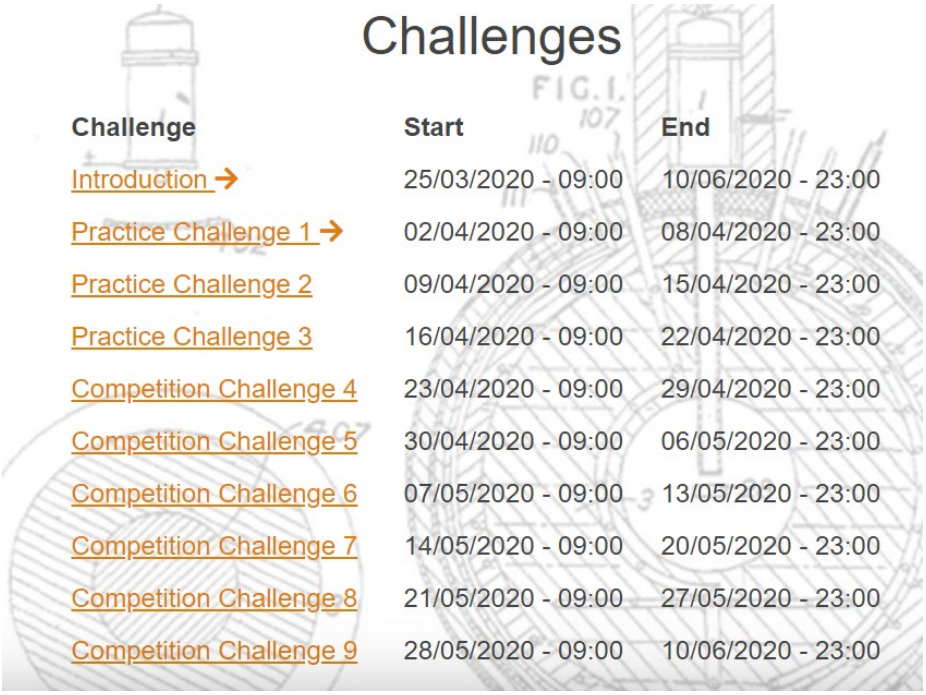
The National Cipher Competition, an annual cryptographic competition run by the mathematics department of the University of Southampton for ages up to 18.

Who can take part?[1]

1. The competition is open to anyone.
2. Entries may be received from individuals or from teams. The teams may be of any size, but we reserve the right to restrict the number of team members listed on the honours board.
3. Team Captains set the team name in their account.
   Teams with more than one member must be set up by a Team Captain who we may contact via email.

Cybites are entering as a team and Anna Matter is the Team Captain so she will be the main person accessing the challenges as only the Team Captain can submit entries on behalf of the team.

Cryptograms are released on a weekly basis every Thursday. Entrants can only access the challenges that are available to submit solutions to when the website is accessed or older challenges where the deadline has passed, and the actual answers are available to be viewed as Teams can no longer submit answers to the relevant challenge.



Challenges

Challenge	Start	End
Introduction →	25/03/2020 - 09:00	10/06/2020 - 23:00
Practice Challenge 1 →	02/04/2020 - 09:00	08/04/2020 - 23:00
Practice Challenge 2	09/04/2020 - 09:00	15/04/2020 - 23:00
Practice Challenge 3	16/04/2020 - 09:00	22/04/2020 - 23:00
Competition Challenge 4	23/04/2020 - 09:00	29/04/2020 - 23:00
Competition Challenge 5	30/04/2020 - 09:00	06/05/2020 - 23:00
Competition Challenge 6	07/05/2020 - 09:00	13/05/2020 - 23:00
Competition Challenge 7	14/05/2020 - 09:00	20/05/2020 - 23:00
Competition Challenge 8	21/05/2020 - 09:00	27/05/2020 - 23:00
Competition Challenge 9	28/05/2020 - 09:00	10/06/2020 - 23:00

Figure 1 Structure of Challenges, Screenshot taken from the NCC website Special Edition

[1] https://www.cipherchallenge.org/information/rules/

As seen above there are 3 practice challenges before the competition releases its proper challenges, but even for the practice challenges, points are awarded, although presumably less points than awarded for the Competition Challenges. The introduction gives context on what the challenges data content should be.

Time Constraints of Competition

Each challenge itself is split into two parts, part a and part b, where points are awarded for part b. Entrants have approximately a week to submit answers for these parts of the challenges. However, there are further time constraints for different amounts of points.

Practice Challenge 3
Deadlines - for B Answers Only

Start	End	Points
16/04/2020 9:00am	17/04/2020 11:00pm	200
17/04/2020 11:00pm	18/04/2020 11:00pm	180
18/04/2020 11:00pm	19/04/2020 11:00pm	160
19/04/2020 11:00pm	20/04/2020 11:00pm	140
20/04/2020 11:00pm	21/04/2020 11:00pm	120
21/04/2020 11:00pm	22/04/2020 11:00pm	100

Figure 2 Point Allocation, Screenshot taken from the NCC website Special Edition

As seen above, if the challenge is completed within the first 2 days, 200 points are awarded with the amount of points decreasing as the time since the challenge was released increases.

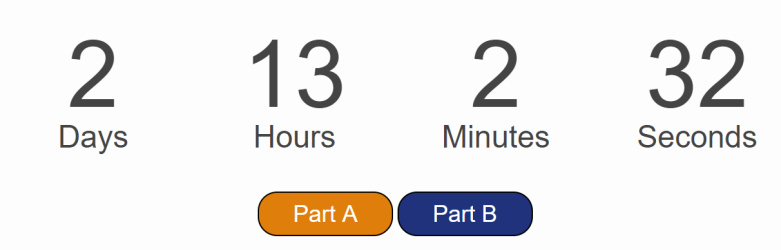
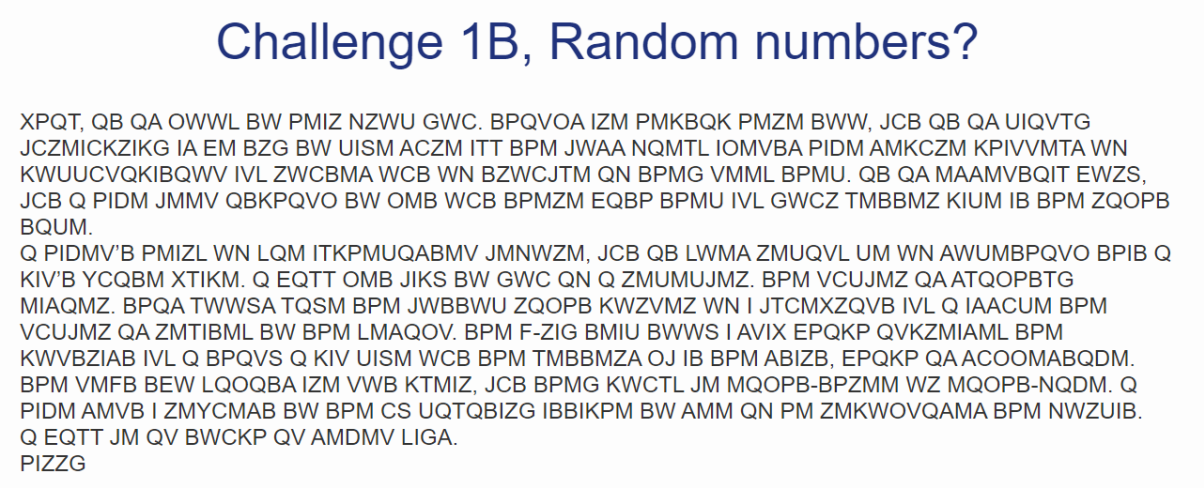


Figure 3 Timer and Parts shown, Screenshot taken from the NCC website Special Edition

The challenges tend to have a length of around 500-1500 characters. An example of a challenge is shown below. The Part A section tends to reveal what type of cipher is used, whereas the Part B generally has only an entry box where the answer to be submitted by the team is located below the data given for the challenge.



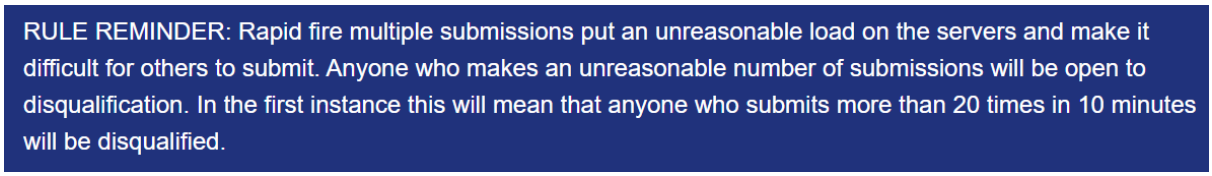
Challenge 1B, Random numbers?

XPQT, QB QA OWWL BW PMIZ NZWU GWC. BPQVOA IZM PMKBQK PMZM BWW, JCB QB QA UIQVTG JCZMICKZIKG IA EM BZG BW UISM ACZM ITT BPM JWAA NQMTL IOMVBA PIDM AMKCZM KPIVVMTA WN KWUUCVQKIBQWV IVL ZWCBMA WCB WN BZWCJTM QN BPMG VMML BPMU. QB QA MAAMVBQIT EWZS, JCB Q PIDM JMMV QBKPQVO BW OMB WCB BPMZM EQBP BPMU IVL GWCZ TMBBMZ KIUM IB BPM ZQOPB BQUM.
Q PIDMV'B PMIZL WN LQM ITKPMUQABMV JMNWZM, JCB QB LWMA ZMUQVL UM WN AWUMBPQVO BPIB Q KIV'B YCQBM XTIKM. Q EQTT OMB JIKS BW GWC QN Q ZMUMUJMZ. BPM VCUJMZ QA ATQOPBTG MIAQMZ. BPQA TWWSA TQSM BPM JWBBWU ZQOPB KWZVMZ WN I JTCMXZQVB IVL Q IAACUM BPM VCUJMZ QA ZMTIBML BW BPM LMAQOV. BPM F-ZIG BMIU BWWS I AVIX EPQKP QVKZMIAML BPM KWVBZIAB IVL Q BPQVS Q KIV UISM WCB BPM TMBBMZA OJ IB BPM ABIZB, EPQKP QA ACOOMABQDM. BPM VMFB BEW LQOQBA IZM VWB KTMIZ, JCB BPMG KWCTL JM MQOPB-BPZMM WZ MQOPB-NQDM. Q PIDM AMVB I ZMYCMAB BW BPM CS UQTQBIZG IBBIKPM BW AMM QN PM ZMKWOVQAMA BPM NWZUIB. Q EQTT JM QV BWCKP QV AMDMV LIGA.
PIZZG

Figure 4 Challenge 1B used as example from NCC Special Edition

Rules of the Competition

Except for the time constraints the main rules consist of the work being the teams own. As well as rapid fire submissions being prohibited and resulting in disqualification from the competition.



RULE REMINDER: Rapid fire multiple submissions put an unreasonable load on the servers and make it difficult for others to submit. Anyone who makes an unreasonable number of submissions will be open to disqualification. In the first instance this will mean that anyone who submits more than 20 times in 10 minutes will be disqualified.

Figure 5 Rule Reminder from NCC Special Edition

Main Problem

About

Although all these ciphers can be done by hand, it takes a significantly larger amount of time to do this as well as it is immensely difficult to crack ciphers without the use of a computer as the key is not known. There is a lot of different ways the data will need to be processed so by using a computer you can reduce the load on the user. All ciphers can be mathematically decomposed as the Caesar and Vigenère cipher are shifts, whereas in the Baconian cipher there is a one to one relationship between each group of characters.

The program cannot make use of online components due to the teams limited resource so it will have to be a standalone program that can run without the use of the internet to decrypt which is how most people would probably approach this challenge.

Computational Methods

Thinking Abstractly & Visualisation

The details that are important are the actual algorithms for encrypting, decrypting and cracking, the inputs for the program as well as the display of the outputs. The details into what colour the window and size of the text and appearance of buttons is unimportant and can be considered after the main components are complete.

Thinking Ahead

The inputs include the selection of which cipher is to be used, which method will be used on the cipher as well the data from the National Cipher Challenge. These are all necessary for the program to work. The key being input will depend on how the team are approaching the challenge as well as the challenge itself.

Thinking Procedurally & Decomposition

The algorithms for the various ciphers can be decomposed into smaller problems and steps, which will occur in the design stage. There will be several decompositions needed especially for the Vigenère cipher as this uses several shifts.

The program can also use an object-oriented paradigm so that the whole program can be split into classes with a single class as a cipher with methods being the processes and the attributes the data input.

Thinking Logically

The user of the program will need menus that flows to the desired selections. This includes two branches for what cipher is used and what process.

In the algorithms themselves, there is likely to be several repeating of various processes, for example, the Vigenère Cipher is like several Caesar Cipher processes applied to the text, meaning it loops around several times, but with different keys. The cracking algorithms will make use of loops and repetition in order to work out the key and cracked text.

Thinking Concurrently

Programming the algorithms to work concurrently from the code itself is beyond my scope of coding, but I will rely on the CPU's inbuilt concurrent processing skills depending on what CPU is used.

Research

Initial Solution Proposed

A program with the following features:

- A new window
- Offline, no need for access to the internet
- Simple selection methods
- Caesar Cipher: Encryption, Decryption, Cracking
- Baconian Cipher: Encryption, Decryption

- Vigenère Cipher: Encryption, Decryption, Cracking
- All algorithms to be completed in a reasonable amount of time. (Under 2 minutes)

Currently available solutions

Cryptii

Cryptii[2] makes use of the internet and has a range of other ciphers available. It doesn't meet the stakeholders needs because it requires internet access but may be used by a developer to check that the answers produced by the offline program in development are correct.

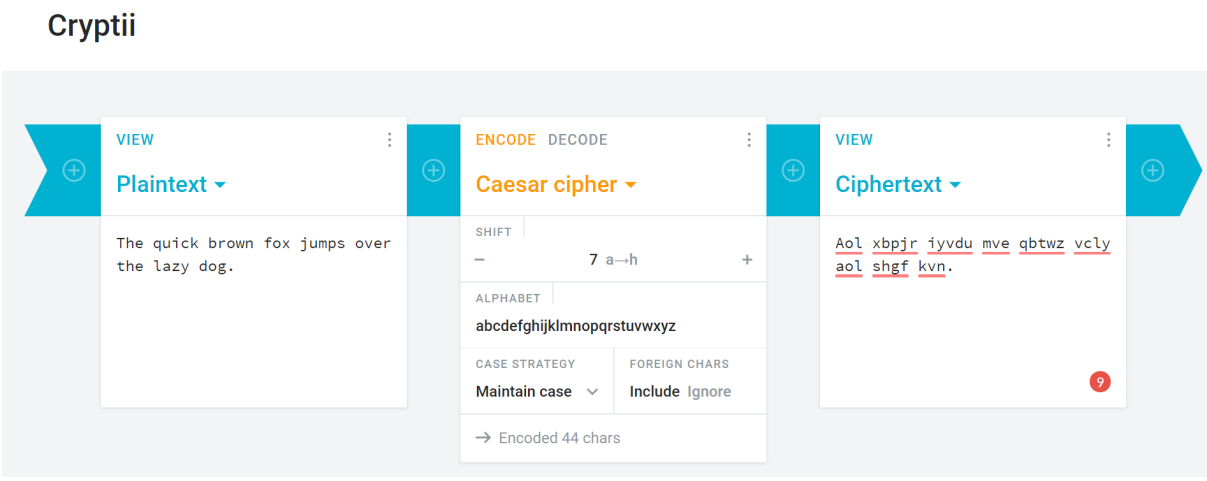


Figure 6 Screenshot of Cryptii

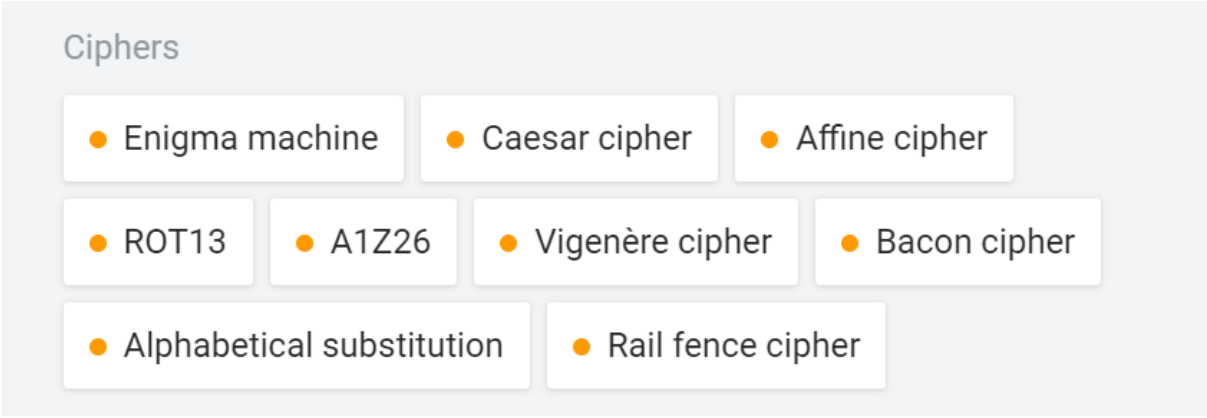


Figure 7 Screenshot of Cryptii

DCode

DCode[3] makes use of the internet and has a range of other ciphers available. It doesn't meet the stakeholders needs because it requires internet access but may be used by a developer to check that the answers produced by the offline program in development are correct.

[2] https://cryptii.com/
[3] https://www.dcode.fr/en

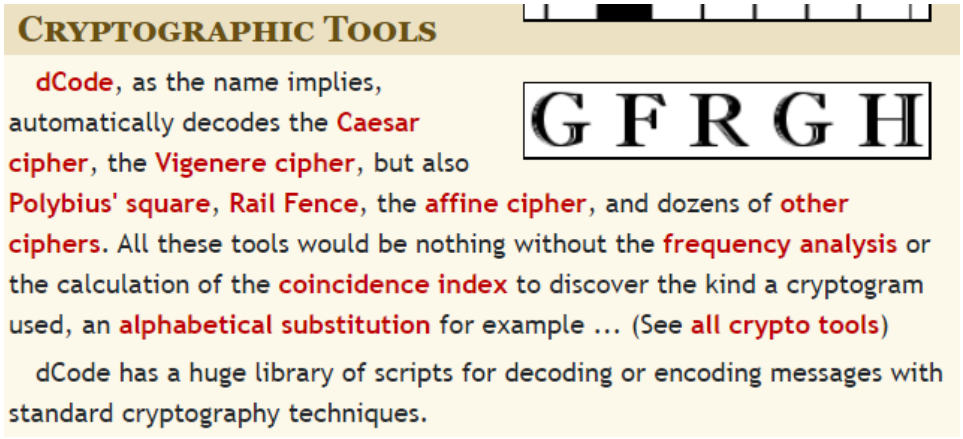


Figure 8 Screenshot from dCode

Figure 9 Screenshot from dCode

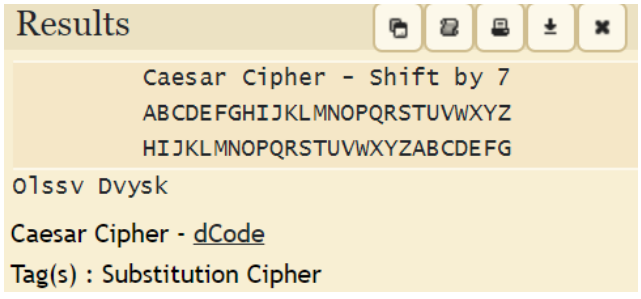


Figure 10 Screenshot from dCode

Caesar Cipher

This is being researched because the stakeholder requires these 4 ciphers to be used successfully in the program

Level of Security: Very low

The Caesar Cipher was used by the ruler Julius Caesar[4] (100BC – 44BC) to encrypt messages on his military orders[5] that were being sent to his generals and information that he did not want exposed to any enemies. It used a method where all the letters in the alphabet will be shifted by the same numerical key and the encrypted message will be made up of the shifted letters which will not make sense if read by someone who has not deciphered it and does not know the key.

The way in which the algorithm can work is that all letters in the alphabet in a certain case can be converted into numerical values with a maximum of 26 and then those number numbers will have the key value added on to it which will create a new numerical values for the cipher text and numbers over 26 will then become their value mod 26. All these numbers will then be converted back into the letters they now correspond to and then when concatenated as they were before being encrypted will produce the cipher text.

E.g. When using a shift of 4: The letter 'A' would become 'D' and the letter 'D' would become 'G'

So, the word 'CAESAR' would become 'F

For displacing one character by the shift key, mathematically[6] as a function:

x being the original plain text character

a being the cipher key

F(x)=(x+a)(mod26)

Converting this function simple python code would be

F(x)= (x+a)%26

(will need more information, will need images code that ex)

[4] https://www.secretcodebreaker.com/history2.html
[5] https://blogs.ucl.ac.uk/infosec/2017/03/03/cryptography-basics/
[6] http://practicalcryptography.com/ciphers/caesar-cipher/

Example:

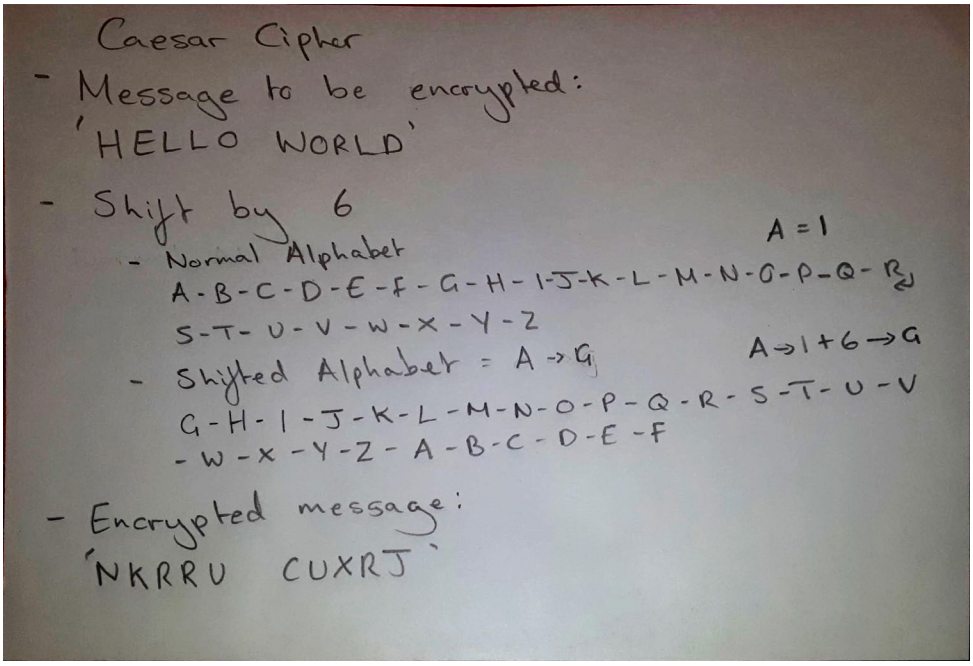


Figure 11 Demonstation of Caesar Cipher (by hand)

Flowchart:

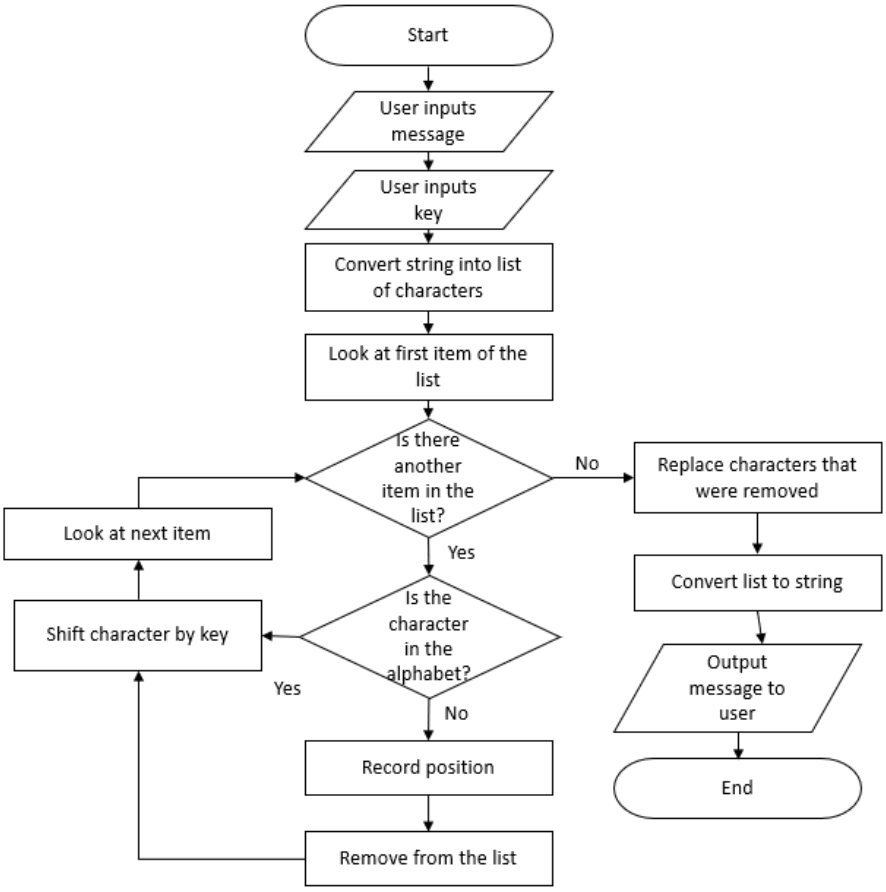


Figure 12 Flowchart demonstrating Caesar Cipher Encryption

Cracking Caesar Cipher

Using frequency analysis can increase the speed in which you can crack the encryption, but ultimately there are only 26 ways in which the message can be encrypted so a brute fore algorithm would also be viable.

Frequency Analysis

Frequency analysis is when the amount of times a letter is recorded and compared against a statistic of how commonly a letter is likely to be used and by comparing these it is possible to work out the key using this information.

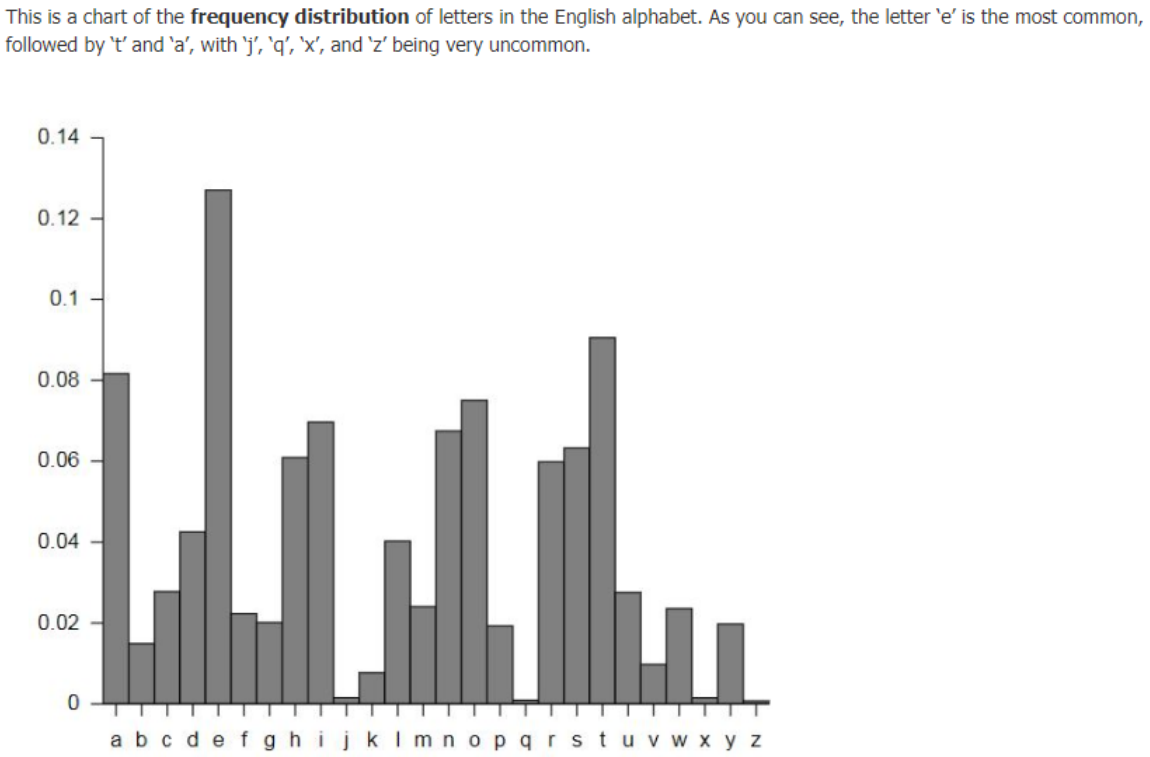


Figure 13 Frequency Table [7]

How does it work computationally?

The spaces in the string will be removed and a list of how many times a letter appears will be created and the letter with the most appearances will be compared against the letter that generally has the highest frequency in sentences determining the shift key for the rest of the ciphertext to be decoded with.

In order to give the most accurate answer to the user, this will be used first and only on request the brute force algorithm will run.

[7] https://learncryptography.com/attack-vectors/frequency-analysis

Amended: There will no longer be a brute force algorithm applied directly on the cipher text. The English checker is used in order to make sure the returned output is English therefore it will not return the output unless it is correct.

Brute Force

This means performing 25 shifts on the cipher text as that is the maximum amount of ways that Caesar cipher text can be encrypted.

Justification:

I will be programming the cracking tool to use frequency analysis to find the most intelligible solution to decrypting the cipher text because this will reduce the amount of time that could be spent looking at the various deciphered versions that come with the brute force method. However, after the frequency analysis test has been conducted, I will give the user the option to use a brute force method, just in case the deciphered text is not in standard English.

Amended: The frequency analysis should automatically return the most intelligible solution in standard English.

Baconian Cipher

What is it?

Level of security: Low

The Baconian Cipher was named after Sir Francis Bacon, the inventor and it is a way of encrypting using a substitution key, where each letter of the alphabet has an equivalent of five characters that it matches up to, this is then concealed[8] within a plain text message to make the fact the message is encrypted harder to work out. In the five characters that each letter corresponds to there will be two different characters, normally 'a' and 'b', starting from A = aaaaa and B = aaaab in a binary format where a=0 and b=1.

From what I've gathered using 5 characters it allows for all the letters of the alphabet to be represented as 5 characters allows for 32 characters to be represented and that is more than what is necessary to represent the alphabet, but sometimes I and J as well as U and V are merged so that only 24 characters are represented so there should be a choice between a 26-character cipher and a 24-character cipher. The amount of characters in the original message is n then the amount of characters in the encrypted message will be 5n. In the end encrypted message there should be exactly 5n characters not including spaces and by using two types of formatting (e.g. capitalization or two different fonts) so say if format 1 is a = lowercase and format 2 is b=uppercase then the message will be a normal message which could say anything but it the formatting will be in correspondence to whether it is an a or b.

The program will need to be able to tell the difference between two formats within the same text and produce two different types of format as well.

[8] http://rumkin.com/tools/cipher/baconian.php

The following tables represent what the original letters are equivalent to in code and binary.

LETTER	CODE	BINARY
A	aaaaa	00000
B	aaaab	00001
C	aaaba	00010
D	aaabb	00011
E	aabaa	00100
F	aabab	00101
G	aabba	00110
H	aabbb	00111
I, J	abaaa	01000
K	abaab	01001
L	ababa	01010
M	ababb	01011
N	abbaa	01100
O	abbab	01101
P	abbba	01110
Q	abbbb	01111
R	baaaa	10000
S	baaab	10001
T	baaba	10010
U, V	baabb	10011
W	babaa	10100
X	babab	10101
Y	babba	10110
Z	babbb	10111

Figure 14 The 24-letter cipher (I/J and U/V merged) [9]

LETTER	CODE	BINARY
A	aaaaa	00000
B	aaaab	00001
C	aaaba	00010
D	aaabb	00011
E	aabaa	00100
F	aabab	00101
G	aabba	00110
H	aabbb	00111
I	abaaa	01000
J	abaab	01001
K	ababa	01010
L	ababb	01011
M	abbaa	01100
N	abbab	01101
O	abbba	01110
P	abbbb	01111
Q	baaaa	10000
R	baaab	10001
S	baaba	10010
T	baabb	10011
U	babaa	10100
V	babab	10101
W	babba	10110
X	babbb	10111
Y	bbaaa	11000
Z	bbaab	11001

Figure 15 The 26 letter cipher

Example:

Encrypting:

Plain text: ‘HELLO WORLD’ n=10

Encoded with 24 letter cipher:

aabbb aabaa ababa ababa abbab babaa abbab baaaa ababa aaabb

[9] https://www.geeksforgeeks.org/baconian-cipher/

The storm is coming Send help please We need reinforcements

Random 50 (=5n) letter phrase: a = **format 1**, b = format 2

The storm is coming. Send help please. We need reinforcements.

Thest ormis comin gSend helpp lease WeNee drein force ments

Encrypted Message: '**The** st**orm** **is** **coming** **Send** **help** **please** **We** **Need** **reinforcements**'

Flowchart:

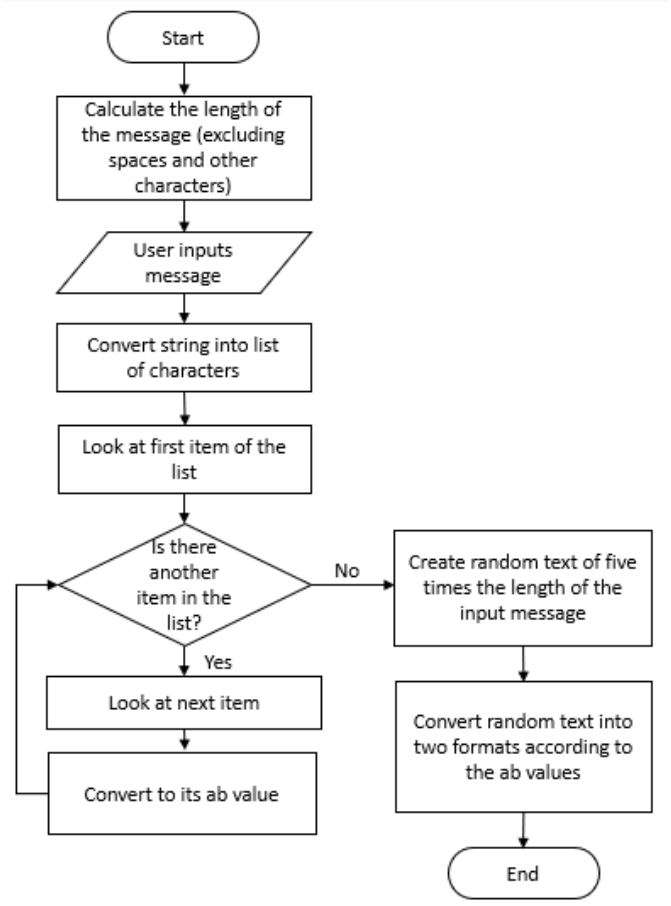


Figure 16 Flowchart demonstating Baconian Cipher Encryption

On further analysis of the Baconian cipher I have decided to restrict it to only a=lowercase, b=uppercase and i=j, u=v because this is the most frequently used format when using the Baconian cipher. It also allows for a slightly more resilient encryption as a computer will not be able to tell if a letter should be 'i' or 'j' but a human can do that easily by sight and decipher the whole text.

Cracking the Baconian Cipher

If the user is aware that this type of encryption is being used, then the deciphering of this text should be relatively simple as the program will need to be able to tell the difference and assign

a or b to this. It will produce two different texts, one the correct deciphered text and the other a scrambled version of the text.

On further analysis of research on the Baconian cipher I have decided to remove any attempt at cracking the Baconian cipher as it is not encrypted using a unique key so decryption should be enough for the user to understand it.

Justification:

I am not going to be programming a way of being able to tell which deciphered text is correct as this should be easy enough for the user to analyse and make a decision on quickly and creating additional code to do this may actually slow the program down unnecessarily for this cipher.

The program will no longer produce multiple deciphered text, but the justification remains partially true as the user should be able to identify if the text is English and choose whether to use the output or not.

Limitations:

I will limit the two formats in the Baconian cipher to uppercase and lowercase letters as this prevents any errors that may occur if the computer were to start analysing the fonts of the text.

As stated before, I also limited the cases further by assigning lowercase to a and uppercase to b.

Vigenère Cipher

The Vigenère Cipher is a polyalphabetic cipher that uses a key word or phrase to encrypt a message. This key phrase is generally unknown to anyone apart from the sender and recipient, but it is generally kept the same in an exchange of information between two people. If the passphrase is shorter than the unencrypted message, then the passphrase will be repeated until it matches the amount of characters in the original message. A flaw in this is that the key generally stays the same for the duration of the session that the exchange takes place, meaning if the cipher is broken then all the messages can be deciphered.

Example:

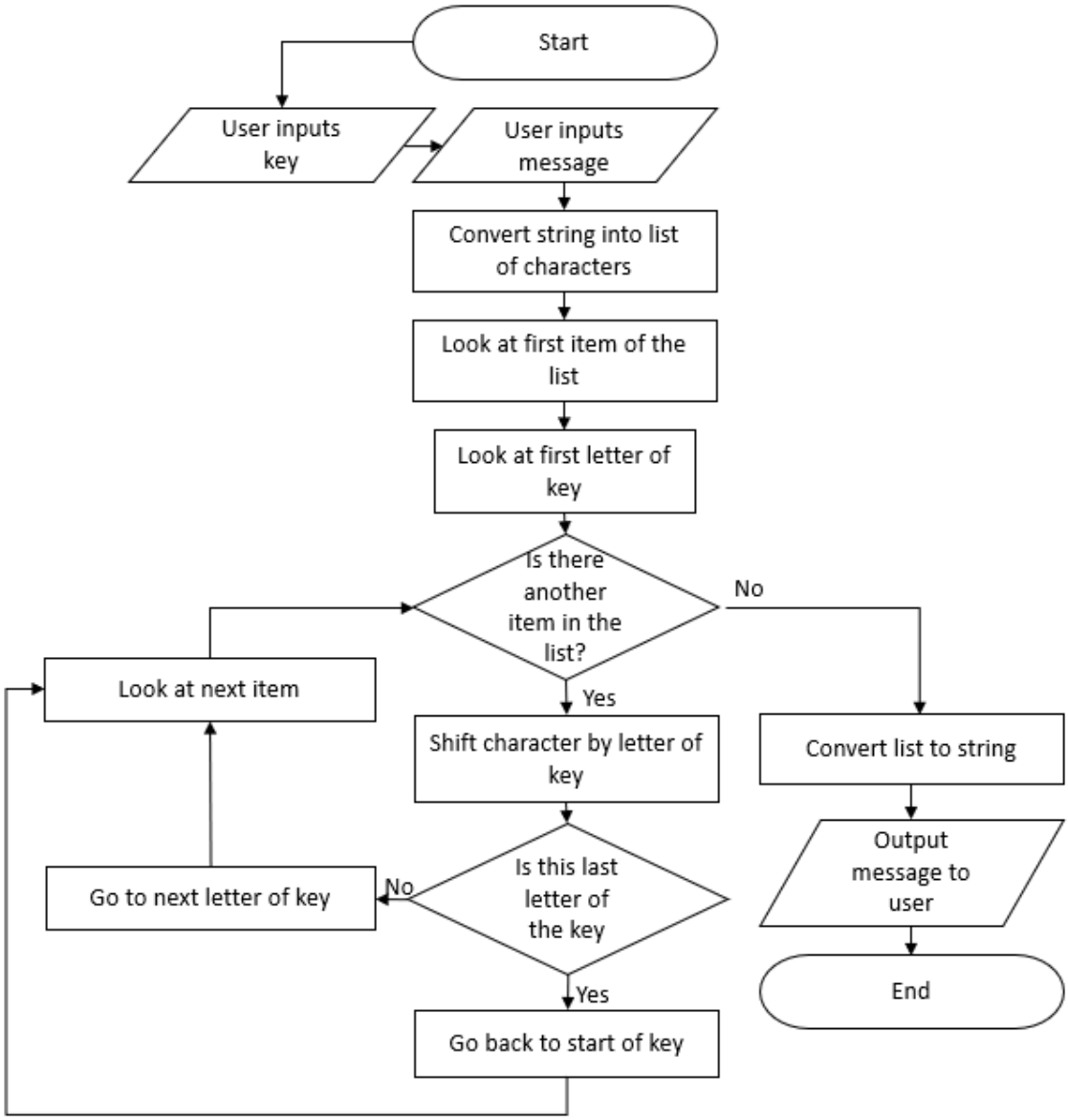
Message to be encrypted: ‘HELLO WORLD’

Key: spam

Actual key: spamspamsp

Encoded text: ZTLXG LODDS

Flowchart:



Cracking Vigenère Cipher

The Vigenère cipher can be approached in two different ways; the dictionary attack and the Kasiski Examination.

Dictionary Attack

The dictionary attack is a brute force method in which a list of English words is used where every word in the dictionary is run through the program as if it is the actual key in order to find the deciphered text.

Kasiski Examination

Where the length of the key is determined in order to use frequency analysis to break the subkeys for each set of characters equal to the length of the main key. It does this through finding repeated sequences within the cipher text and the tries to work out by what factor

these subkeys are spaced and seeing if the whole cipher texts divides into this and then using frequency analysis to decode the ciphertext fully.

It starts with the text being analysed for repeated groups of 3 characters. The length of characters between these repeated groups is then determined and the common factors within these numbers are found and a dictionary of these are found.

The most common factors will be the most likely to be the key, the most common factor will be used as the key length and a brute force algorithm will be applied.

After programming this I realized that one would often be the most common factor, which would reflect Caesar cipher instead of the Vigenère cipher, so I removed 1 as a key length.

After programming this I realized that the brute force algorithm was taking a mass amount of time and didn't seem to be returning anything in a realistic amount of time, so I researched the Kasiski Analysis method further.

On further examination of the Kasiski test, I realize I missed out a crucial stage which was a character frequency analysis stage. The further research is stated below.

Upon researching the frequency analysis stage of the Kasiski examination. I realized I was unable to use my previously programmed frequency analysis from the Caesar cipher because of the nature of that module, but I was able to abstract and use certain elements of it like a dictionary being produced.

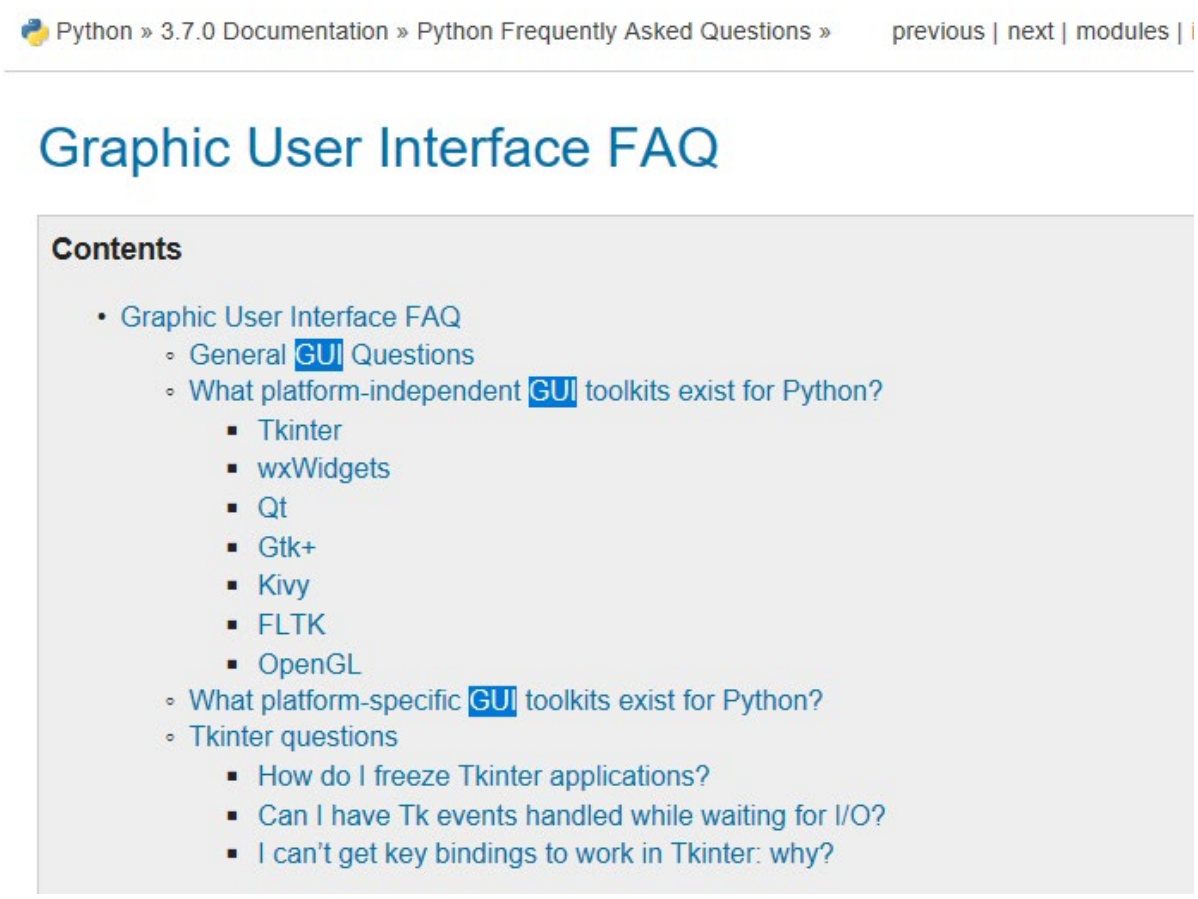
The aim of this frequency analysis was to create separate strings of every letter of the key length so if a key was 3 letters long there would be a string for 3n, 3n+1, 3n+2 and then I developed my solution like my previous frequency analysis so I matched the most frequent letter up to one of the 6 most frequent letters used in the alphabet and managed to gain a selection of possible values of each letter of the key and then put them together to produce several keys. Using these keys, it decrypts the text with the key and uses to English checker to output the correct decryption.

Justification:

I will make use of both cracking algorithms, but I will set the program to use the Kasiski algorithm since the dictionary attack relies on the key being an English word, but a dictionary attack is performed regardless as it is part of the English checker program. This is required by the stakeholder to complete the competition challenges.

GUI

This is being researched because the stakeholder requires the program to be interactive in a basic way where they can use buttons and simple input boxes to navigate within the program.


Python » 3.7.0 Documentation » Python Frequently Asked Questions » previous | next | modules |

Graphic User Interface FAQ

Contents

- Graphic User Interface FAQ
  - General GUI Questions
  - What platform-independent GUI toolkits exist for Python?
    - Tkinter
    - wxWidgets
    - Qt
    - Gtk+
    - Kivy
    - FLTK
    - OpenGL
  - What platform-specific GUI toolkits exist for Python?
  - Tkinter questions
    - How do I freeze Tkinter applications?
    - Can I have Tk events handled while waiting for I/O?
    - I can't get key bindings to work in Tkinter: why?

Kivy

Kivy is a GUI Library that is cross-platform, supporting multiple desktop operating systems such as Windows, MacOS and Linux as well as mobile devices such as Android and iOS. Kivy is a free, open source software which is distributed under the MIT licence agreement. It can be written with python, but it has less support than more popular GUI interfaces.

Tkinter

Tkinter is the standard python GUI toolkit and the most popular. It is installed automatically with the standard installation of Python on operating systems, Windows, MacOS and Linux. It is free and since it is installed with python, the most easily accessible GUI available for python. It also has the most support available as it is the most popular GUI interface for python.

Examples:

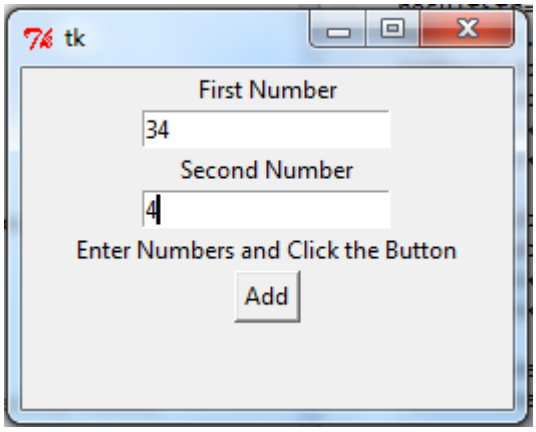


Figure 17 https://bakedcircuits.wordpress.com/2013/08/13/beginning-gui-programming-in-python/

PyGUI

Another crossplatform framework used for creating graphical interfaces to interact with users. It is the simple and lightweight of the GUI librarys as it uses an Application Program Interface that aligns with python. However there is a lack of support for this interface because while it is the simplest to use, it is not the most popular.

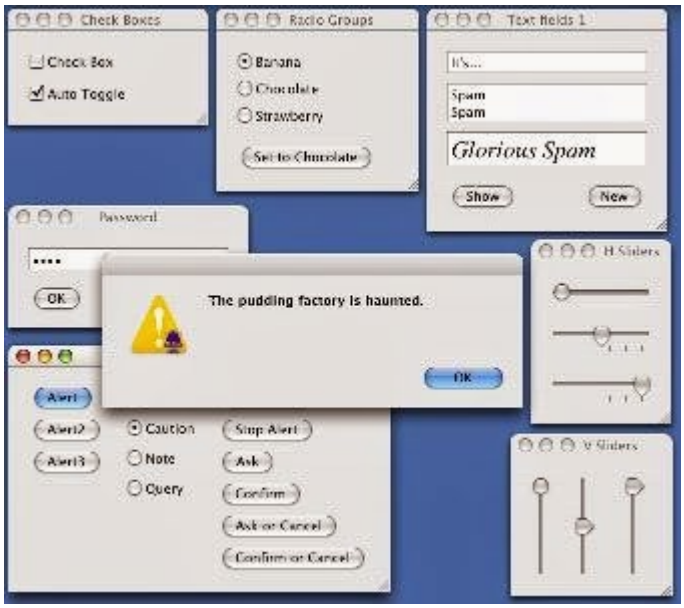


Figure 18 https://python-catalin.blogspot.com/2011/03/pygui-24-is-available-from.html

Conclusion:

I will be using Tkinter as there is a lot of support for programmers with little experience with GUI interfaces. As well as there is an abundance of modules that work with Tkinter with fewer compatibility issues. If I used a simpler GUI toolkit there is a risk that I may encounter there is a feature not available that I need to use.

Final Solution Proposed

The Caesar cipher is computable as it an monoalphabetic substitution cipher where the letters of the message shift by the key = n. Cracking the Caesar cipher is also a soluble and tractable problem as there are maximum 25 shifts that the message can be deciphered into, which

should not take too much time for a letter and use of frequency analysis can reduce time and present the best solution for the deciphered text.

The Baconian cipher is computable as it is a monoalphabetic substitution cipher where the letters are replaced using a key that can be defined. One problem is that random text will need to be generated, but this should be manageable as the text doesn't necessarily have to make sense as the text itself holds no semantic value. Decrypting the Baconian cipher can also be done computationally because one challenge will be getting the decrypting algorithm to recognise the difference between uppercase and lowercase letters, which can be done using the ASCII table values as uppercase and lowercase letters are already defined differently.

The Vigenère cipher is a polyalphabetic substitution cipher and for encryption and decryption is a computable as it is just a series of shifts depending on an alphabetic key that will loop until the whole piece of text has been encrypted/ decrypted. Cracking the Vigenère cipher will be more of a challenge and the ways of doing this include using the Kasiski method to deduce the most probable length of key and then further deduce what the key is, through a dictionary attack or brute force attack. The algorithm will then decrypt the text using the found key.

Different levels of abstraction of key inputs will be needed in order to convert the input text into formats where the text can be converted using encryption, decryption or cracking successfully. For most ciphers, in encryption and decryption it is possible to leave any spaces, numbers or special characters as they can be ignored in the text, however for cracking it is likely that any numbers or special characters will need to be removed in order to successfully crack the text. The spaces will need to be removed depending on what type of cracking is used. For the English checker, spaces are necessary for the algorithm to be able to differentiate between words so in some cases, placeholders will need to be recorded in order to return the spaces if they have been removed prior to the English checker algorithm being run.

Essential Features: (general features)

The program must run in a separate window with a clear GUI so that the team can switch between the cipher program and an internet browser because the challenges will be accessed using the internet. This is because the stakeholder needs it.

There must be a small menu so that the team can select what cipher they would like to use and what process they wish to carry out. This is because of the nature of the task needing a menu to offer different options.

It must be simple to use with clear inputs and outputs so that the program can be used efficiently because the stakeholder has asked for this.

Limitations

I will be put a limit of 1200 characters so that the program runs efficiently. This allows for roughly two paragraphs of plain text to be entered and it shouldn't take too long to encrypt, decrypt and crack the code.

This limitation was removed because the Vigenère Cipher Cracking sometimes requires a higher amount of characters in order to work as well as the competition sometimes has higher than 1200-character entries.

For the Baconian cipher I will only be using capital letters and lowercase letters so the program will have to recognise the difference between these characters.

Further Limitations were added so that the cases are locked a=lowercase and b=uppercase.

I will be assuming that any numbers and characters put into the coded message are not encoded.

Physical and Device limitations include the fact that only two students out of the four go to schools where devices such as laptops can be brought in. So, the setup of the team will be that two people will be working on paper to attempt to solve the challenge and two people will be working on laptops to access the challenge and the program. Due to limited resources the team want to limit internet access to only accessing the NCC competition challenges and so the program shouldn't use any constantly accessed online components.

As well as they are limited by the rules of the National Cipher Challenge stated above.

System Requirement:

Hardware

A laptop or desktop computer that can run python and its modules. While a normal processor will execute the code fine, a faster processor is preferred so that the program can produce results faster. The peripherals that are generally used to work a computer such as a monitor, a mouse and a keyboard are all that is required other than the computer itself.

Software

2.4. Miscellaneous

To easily use Python scripts on Unix, you need to make them executable, e.g. with

```
$ chmod +x script
```

and put an appropriate Shebang line at the top of the script. A good choice is usually

```
#!/usr/bin/env python
```

which searches for the Python interpreter in the whole `PATH`. However, some Unices may not have the **env** command, so you may need to hardcode `/usr/bin/python` as the interpreter path.

To use shell commands in your Python scripts, look at the `subprocess` module.

Figure 19 Unix Executable Code for Python

The operating system must be Windows, Macintosh or Unix. Preferably Windows or Macintosh as an extra piece of code must be executed in the command prompt in order to use python scripts on Unix[10].

The minimum requirements include Python 3.7 as this must be installed on the computer along with the modules; tkinter, random and string, which normally all are installed when Python itself is installed.

The solution has been coded using Visual Studio Code IDE, but this doesn't have to be installed on the computer in order to run the program.

Most of the modules used in the program are installed with Python 3, but I have used one program that was not stated in initial analysis.

Using pip, the module pyperclip needs to be installed in order to use to copy button in the GUI.

Solution Developed on

This solution was developed on:
Visual Studio Code

Model Name:	HP Pavilion x360
Operating System:	Windows 10
Processor Type:	Intel Pentium
Processor Speed	2.30 GHz
Storage Type:	Hard Disk Drive
Storage Capacity:	1TB
RAM Size:	8GB

Model Name:	Lenovo IdeaPad C340
Operating System:	Windows 10
Processor Type:	AMD Ryzen 3 3200U
Processor Speed	2.60 GHz
Storage Type:	Solid State Drive
Storage Capacity:	128GB
RAM Size:	8GB

Success Criteria

No.	Criteria	Proof	Justification/ Reason
1	The program should run in a separate window using a GUI.	Screenshot of the GUI running.	The user requires the ability to interact with the program.
2	The program should bring up a menu with buttons in which the user can choose which cipher they are using.	Screenshot of the menu.	The stakeholder requires the choice on what cipher they intend to use.

[10] https://docs.python.org/2/using/unix.html

3	The program should then produce another menu asking the user whether they want to encrypt, decrypt and crack.	Screenshot of the menu.	The stakeholder requires the choice on what they intend to do with the chosen cipher.
4	The program should be capable of successfully encrypting a message using Caesar cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
4i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
4ii	The program must be able to validate the key format for the Caesar Cipher.	A screenshot of an error message when the wrong key is input.	This is due to both the nature of the task needing this data to be correct.
4iii	The program must be able to correctly encrypt using Caesar Cipher regardless of where it is output.	A screenshot of the correct encryption output.	This is what the stakeholder requires.
4iv	All results of encryption should be displayed in the GUI.	A screenshot of the GUI after encryption option is chosen.	This is to show the user the result because that is what the user requires.
5	The program should be capable of successfully decrypting a message with the key using Caesar cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
5i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
5ii	The program must be able to validate the key format for the Caesar Cipher.	A screenshot of an error message when the wrong key is input.	This is due to both the nature of the task needing this data to be correct.
5iii	The program must be able to correctly decrypt using Caesar Cipher regardless of where it is output.	A screenshot of the correct decryption output.	This is what the stakeholder requires.
5iv	All results of decryption should be displayed in the GUI.	A screenshot of the GUI after decryption option is chosen.	This is to show the user the result because that is what the user requires.
6	The program should be capable of executing a brute force algorithm as a means of cracking the Caesar cipher.	A screenshot of all the possible shifts for a string of code will be shown.	This and/or criteria 7 is what the stakeholder requires.
6i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.

6ii	The program must be able to correctly crack the Caesar Cipher regardless of where it is output.	A screenshot of the correct cracked text output.	This is what the stakeholder requires.
6iii	All results of cracking should be displayed in the GUI.	A screenshot of the GUI after crack cipher option chosen.	This is to show the user the result because that is what the user requires.
7	The program should be capable of executing a frequency analysis algorithm as a means of cracking the Caesar cipher.	A screenshot of the result given by the algorithm will be given alongside the key that it has found.	This and/or criteria 6 is what the stakeholder requires.
7i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
7ii	The program must be able to correctly crack the Caesar Cipher regardless of where it is output.	A screenshot of the correct cracked text output.	This is what the stakeholder requires.
7iii	All results of cracking should be displayed in the GUI.	A screenshot of the GUI after crack cipher option chosen.	This is to show the user the result because that is what the user requires.
8	The program should be capable of successfully encrypting a message using the Baconian cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
8i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
8ii	The program must be able to correctly encrypt using Baconian Cipher regardless of where it is output.	A screenshot of the correct encryption output.	This is what the stakeholder requires.
8iii	All results of encryption should be displayed in the GUI.	A screenshot of the GUI after encryption option is chosen.	This is to show the user the result because that is what the user requires.
9	The program should be capable of successfully decrypting a message using the Baconian cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
9i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
9ii	The program must be able to correctly decrypt using Baconian Cipher regardless of where it is output.	A screenshot of the correct decryption output.	This is what the stakeholder requires.

9iii	All results of decryption should be displayed in the GUI.	A screenshot of the GUI after decryption option is chosen.	This is to show the user the result because that is what the user requires.
10	The program should be capable of successfully encrypting a message using the Vigenère cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
10i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
10ii	The program must be able to validate the key format for the Vigenère Cipher.	A screenshot of a function returning and displaying the text that has been received.	This is due to both the nature of the task needing this data to be correct.
10iii	The program must be able to correctly encrypt using Vigenère Cipher regardless of where it is output.	A screenshot of the correct encryption output.	This is what the stakeholder requires.
10iv	All results of encryption should be displayed in the GUI.	A screenshot of the GUI after encryption option is chosen.	This is to show the user the result because that is what the user requires.
11	The program should be capable of successfully decrypting a message with the key using the Vigenère cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.
11i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
11ii	The program must be able to validate the key format for the Vigenère Cipher.	A screenshot of a function returning and displaying the text that has been received.	This is due to both the nature of the task needing this data to be correct.
11iii	The program must be able to correctly decrypt using Vigenère Cipher regardless of where it is output.	A screenshot of the correct decryption output.	This is what the stakeholder requires.
11iv	All results of decryption should be displayed in the GUI.	A screenshot of the GUI after decryption option is chosen.	This is to show the user the result because that is what the user requires.
12	The program should be capable of executing a cracking algorithm for the Vigenère cipher.	A screenshot of a test string of text input and the output displayed.	This is what the stakeholder requires.

12i	The subprogram for this needs to run in under 2 minutes.	Time output in the terminal	This is a stakeholder requirement.
12ii	The program must be able to correctly crack the Vigenère Cipher regardless of where it is output.	A screenshot of the correct cracked text output.	This is what the stakeholder requires.
12iii	All results of cracking should be displayed in the GUI.	A screenshot of the GUI after crack cipher option chosen.	This is to show the user the result because that is what the user requires.
13	The program should have an English checker	A piece of random standard English text will be input, and the result will be screenshotted.	This will be used by the program and the user to access the success of decryption and cracking.
13i	Measuring the quantity of English in the given text for the length of the given text.	A screenshot of the code that specifies the ratio being created.	This is due to the nature of the task
13ii	Must return some sort of false or negative if the text input is not English.	A screenshot of the result of a non-English piece being input	This is due to the nature of the task as if it always returns positive then the function is not useful.
14	Output of at least 1000 characters can be shown in GUI so to prove that all of output is visible to the user.	A screenshot of an encrypted/decrypted piece of text length more than 1000 characters.	This is due to the nature of the task.
15	An error message must be shown if a cracking algorithm was not able to successfully produce a result.	An error message will be coded, and erroneous data will be entered so that the error message can be screenshotted.	This is due to the nature of the task.
16	The program should have functions to navigate the program such as reset buttons and an exit button.	A screenshot of the GUI at the main menu and the second menu will show the exit button.	This is due to the nature of the task as the user could select the option wrong.
16i	There should be reset buttons for each function.	A screenshot of the reset buttons.	This is due to the nature of the task and needing to reset the information in the cipher.
16ii	There should be a way of switching to different cipher.	A screenshot of these buttons.	This is due to the nature of the task

			and needing to switch from cipher to cipher.
16iii	There should be a way to exit the program.	Screenshot of the button.	This is due to the nature of the task.
17	The program should have some way of being able to copy the output into another window.	A screenshot of the developed solution and a screenshot of the same data copied and pasted outside of the program.	This is one of the stakeholder's requirements.

Overall, this set of success criteria should help me achieve an end product that meets the stakeholder's requirements as it is a decomposition of the stakeholders needs.

Design

Programming Paradigm

Code should be in an object-oriented format. This should involve the use of classes.

This has been chosen due to the nature of the task as it increases readability and it was determined that the code should be object oriented as it cooperates more when interacting with the GUI.

Planned Objects:

GUI
ShowCaesar()
ShowBaconian()
ShowVigenere()

Caesar
Key
Text
Encrypt()
Decrypt()
Crack()

Baconian
Text
Encrypt()
Decrypt()

Vigenere
Key
Text
Encrypt()
Decrypt()
Crack()

Overview of System

Main Menu & Format

I have decomposed the overall program into the different ciphers and then furthermore the basic processes that will be used so it is clear to the stakeholder as to how the program is to

be used. The first decomposition of the problem has been splitting the ciphers up into different subprograms and this is, so the user's menu choices are clear for the stakeholders to follow.

Success Criteria Aim:

1, 2, 14, 16

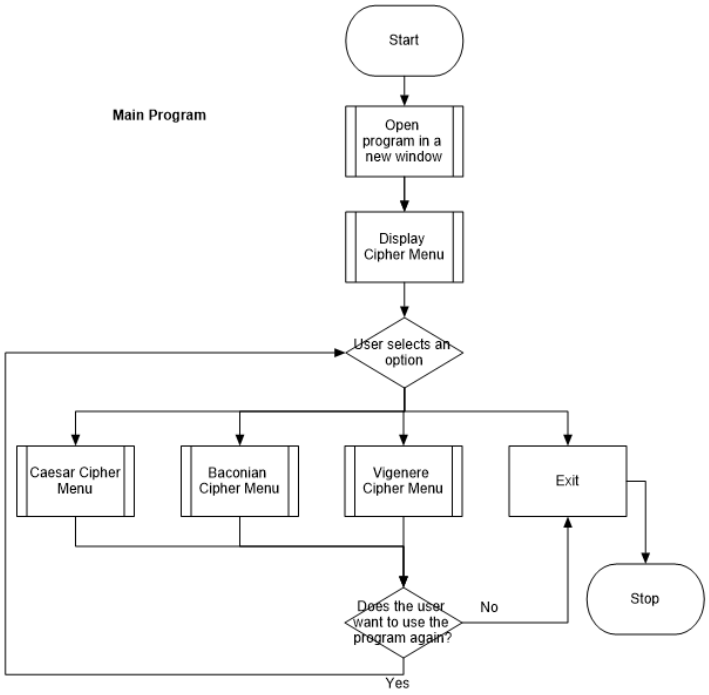


Figure 20 First Menu

Using a flowchart, I am attempting to demonstrate how the main program will work for the initial user interface. The program will open in a new window using Tkinter and using the buttons that will represent each of the subprograms in the 'Main Program' above they will be able to navigate the through the subprograms as demonstrated below. This program will be contained as a function called cipherMenu.

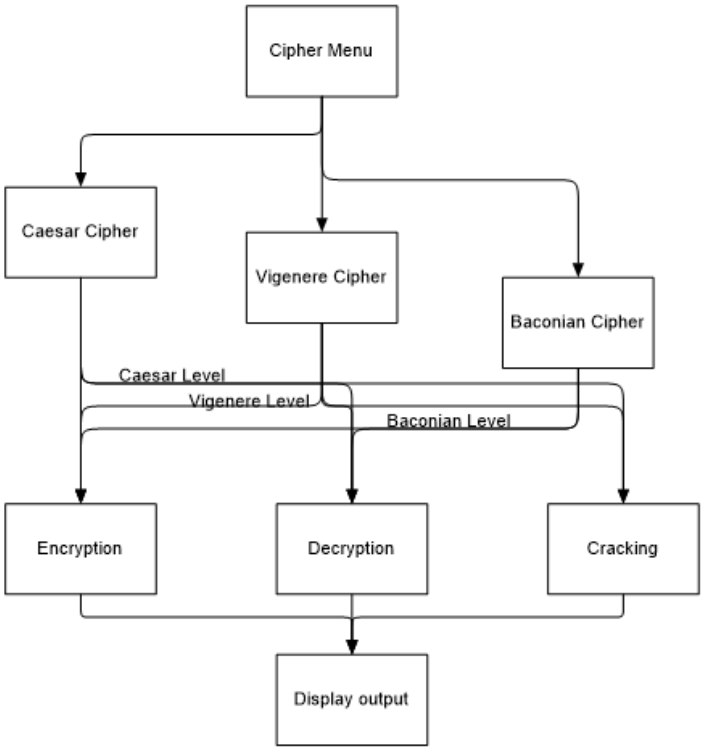


Figure 21 Second (Process) Menu

Figure 22 Start-Up screen

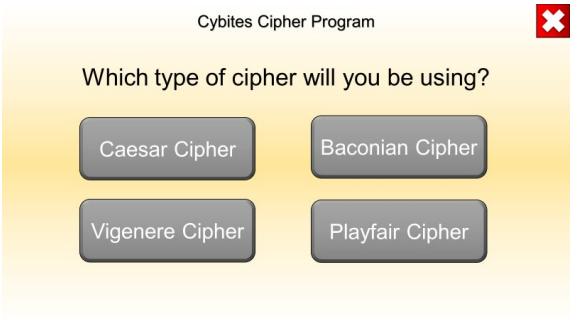


Figure 23 Cipher Menu

I have decided to use light yellow as a background colour as it is a warm colour and improves the readability of long pieces of text. This is after I briefly researched and found a study[11]

[11] https://www.cs.cmu.edu/~jbigham/pubs/pdfs/2017/colors.pdf

conducted by a Human-Computer Interaction Institute which found that warm colours as a background has significant positive impacts on the readability of pieces of text, especially for people with dyslexia, but also for people who don't have dyslexia.

Using a GUI, I am providing buttons as a form of input to make the program as easy to use as possible which is what the stakeholder requires. It also eliminates the possibility of erroneous inputs being entered where options are provided.

The planned dimensions of the GUI 800x600 px, but this is subject to change if text output of 1000 doesn't fit in the frame.

Process Menus

Caesar Cipher Menu

Success Criteria Aim:

3, 4, 5, 6, 7

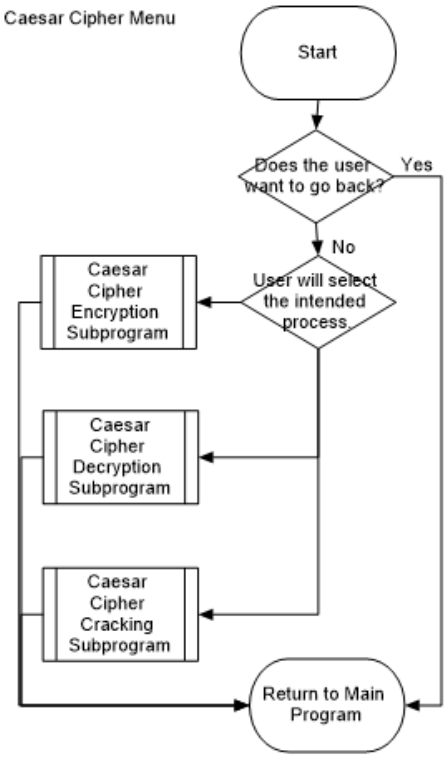


Figure 24 Caesar Cipher Menu

Good Background Colors for Readers: A Study of People with and without Dyslexia
Luz Rello, Jeffrey P. Bigham, Human-Computer Interaction, Carnegie Mellon University

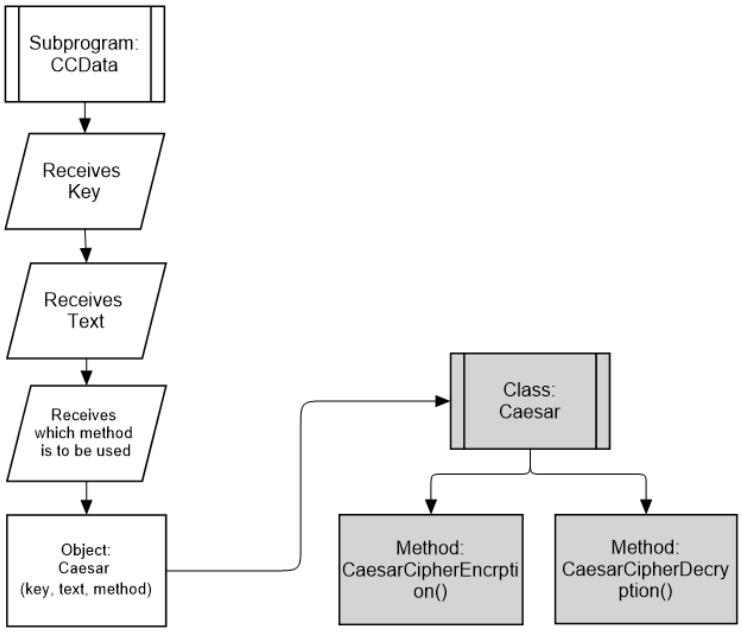


Figure 25 Caesar Cipher (En/De)cryption Diagram

Baconian Cipher Menu

Success Criteria Aim:

3, 8, 9

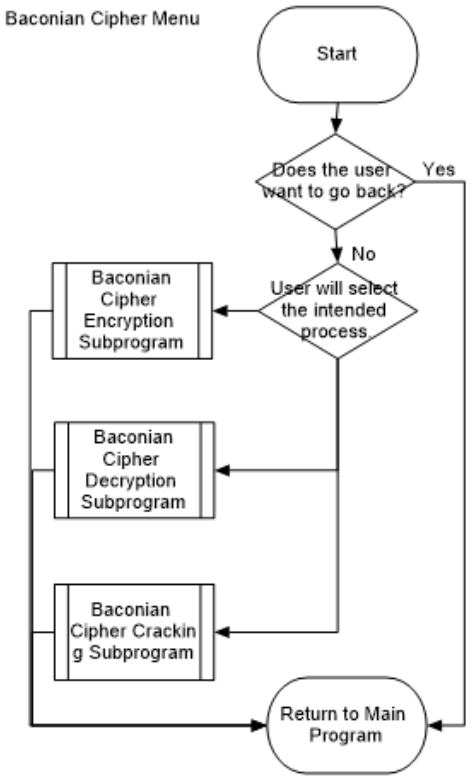


Figure 26 Baconian Cipher Menu

Vigenère Cipher Menu

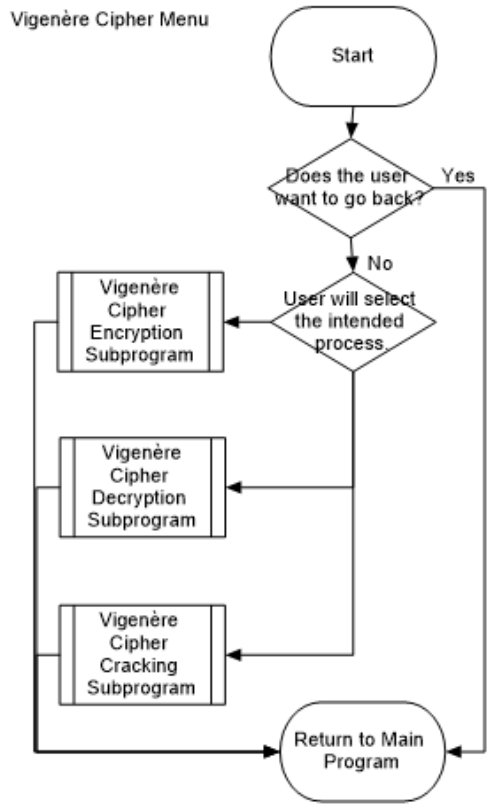


Figure 27 Vigenère Cipher Menu

Summary

The subprograms referenced in these flowcharts will also represent buttons which will direct the screens where they input set the required information. After they have entered the information required, they will then click a button to confirm that the information is correct and the cipher process for the selected cipher will be carried out using the data they have input. This is because the stakeholder requires this as a necessity otherwise the program is redundant because the user will not be able to choose a process to pursue.

Below I have produced images for what the GUI should roughly look like for the overview of the system. This does not yet include the algorithms for the ciphers as they will be given separately later.

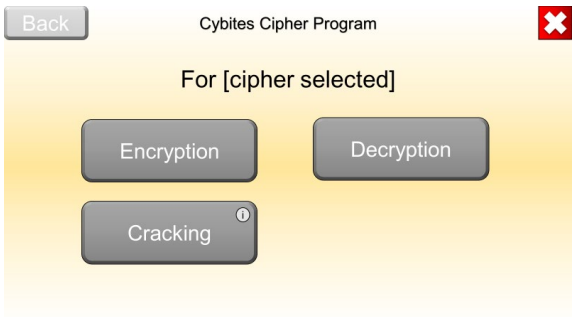
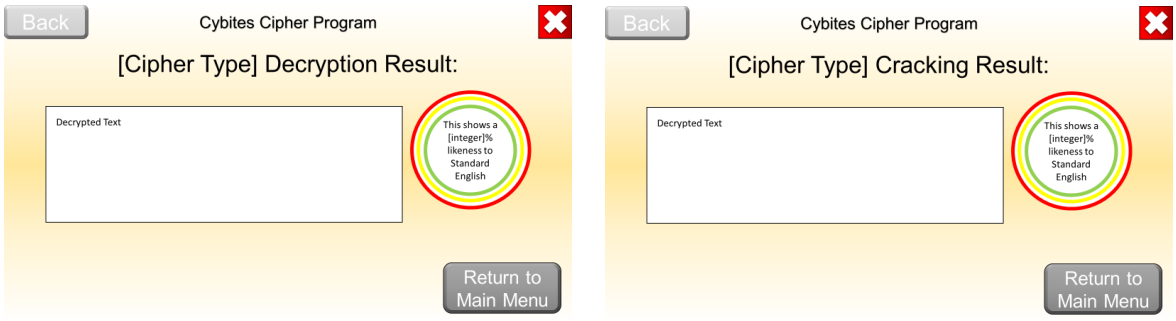


Figure 28 Process Selection Screen

The user will then select from the process they intend to use the selected cipher for.

I will be keeping the GUI simplistic and making use of buttons and various types of text boxes as means of the user interacting with the algorithms. The user may want to exit at any time, but the exit button will be provided by TKinter.

Output Screens



Below are the GUI representations of what the decryption and cracking output screens will look like.

Figure 29 Decryption and Cracking Screen

I have kept the decryption and cracking result screens quite simple by having the decrypted text in a text box, so it is clear what the user needs to read. The circle to the right side will indicate how much of the text is English according to a dictionary and the circle will change colour dependent on the percentage given to it from the English checker. The result of how close to English being returned was specified in the success criteria and the coloured circle just makes the program more user-friendly as some people respond to colours as indicators better than numbers.

The algorithm for this can be found below in English Checker.

Algorithms

English Checker

Success Criteria Aim:

13(i, ii)

Will perform a dictionary attack on each word that is passed through the function. It will also produce a coloured ring to help visually represent how close to english the text is.

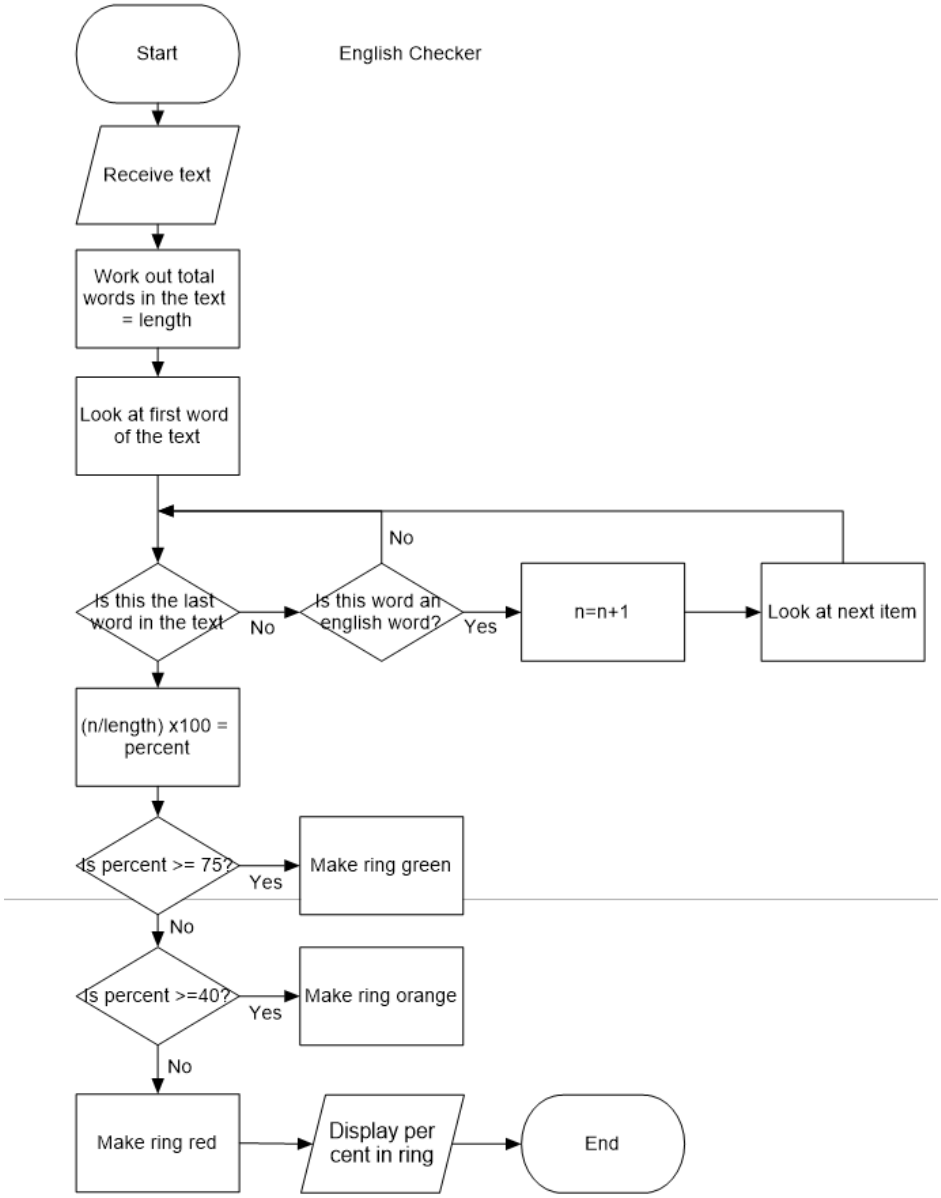


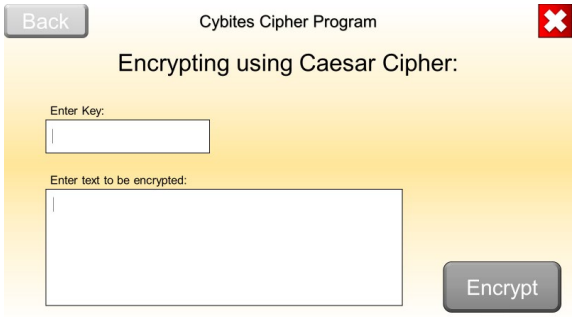
Figure 30 English Checker Flowchart

This changed so that there is no coloured ring as it is not required and will only produce the correct output.

Caesar Cipher Encryption

Success Criteria Aim:

4



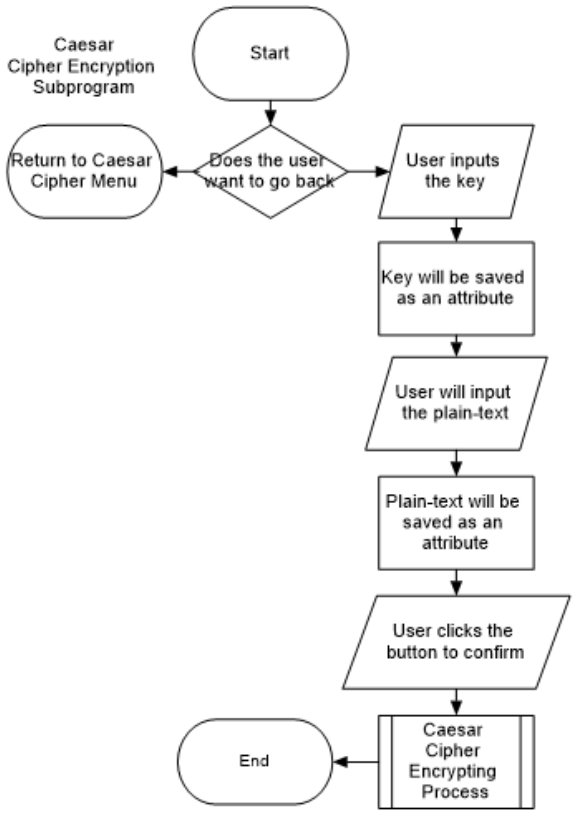
Screen that the user will input the key and text to be encrypted.

The following subprograms demonstrate how the data that has been entrered will be used to encrypt the data.

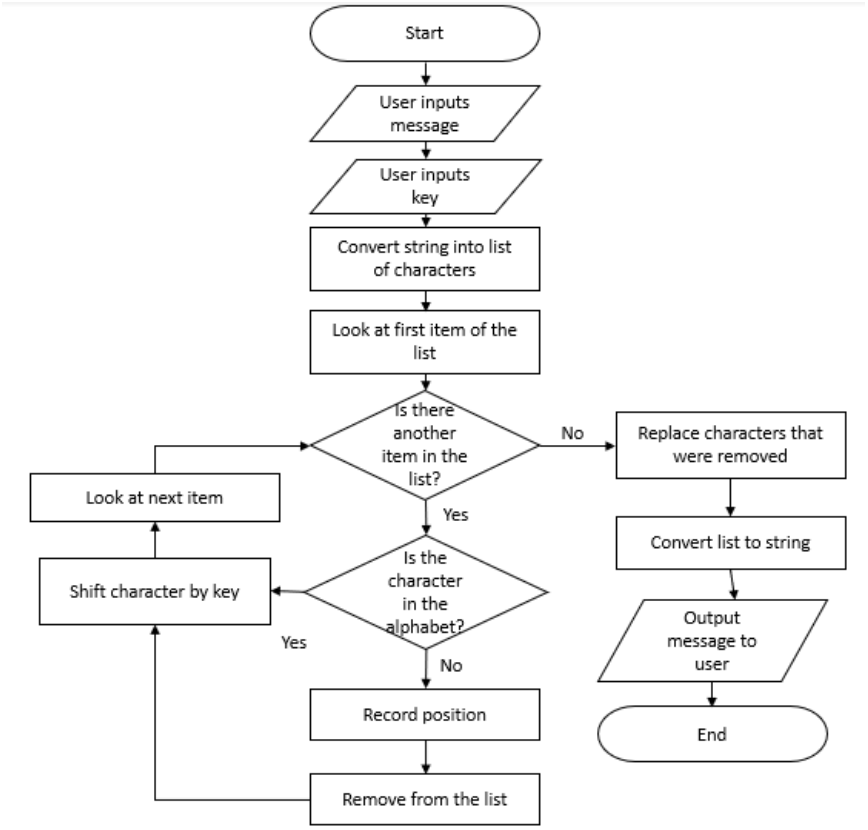
For each character in the string of text given by the user, the type of the character will be determined and if the character is alphabetical then it will convert the letter to its ordinal equivalent.

1) Look at first character in the given string, if it is alphabetical, turn the letter into its ASCII ordinal value and add to a list, if not ignore and move on to the next character.
2) With the new list, add the value of the key to each item in the list and MOD by 26
3) For each item in the list convert it back into its alphabetical equivalent.
4) Make the list a string, adding in any removed characters.
5) Display string to the user.

The key will be added to the ordinal value and then if the value of the key added to the given ordinal letter



This flowchart represents the flowchart of the data needed to from the user to perform the Caesar cipher encryption.

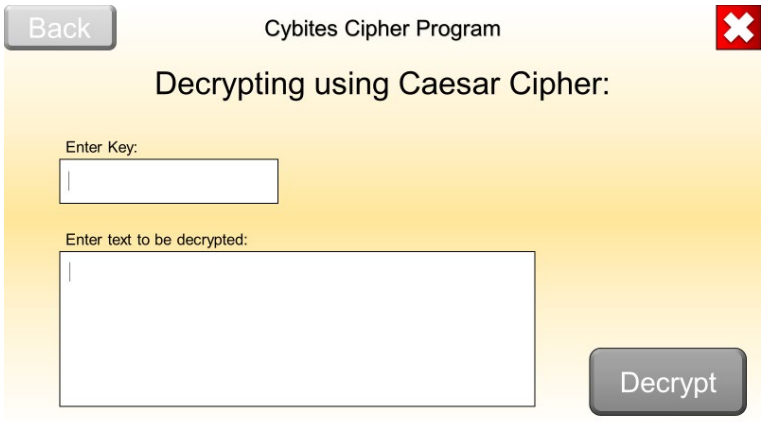


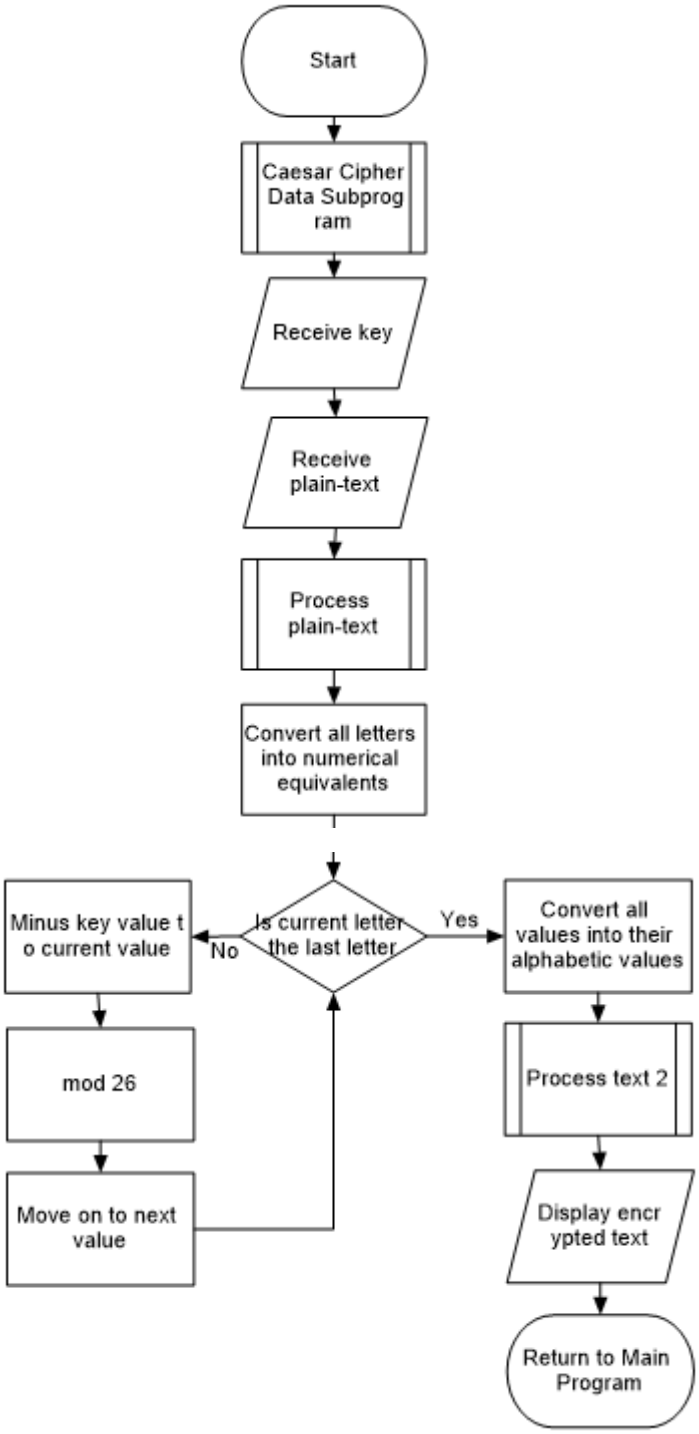
```
key=input(int("Enter key: "))
plaintext=input("Enter text to be encrypted: ")
public function CaesarCipherEncrypt(key, plaintext)
      textlist = convert plaintext to list of characters
      encryptlist = []
      for i = 0 to (textlist.length)
            if list[i] == "abcdefghijklmnopqrstuvwxyz" then
            //If equal to any one of these characters
                  a=convert value to ordinal values
                  a=a+key
                  a=convert back to alphabetical values
                  a.amend to encryptlist
            next i
      encrypttext= convert encryptlist to a string
      return encrypttext
end function
```

Caesar Cipher Decryption

Success Criteria Aim:

5

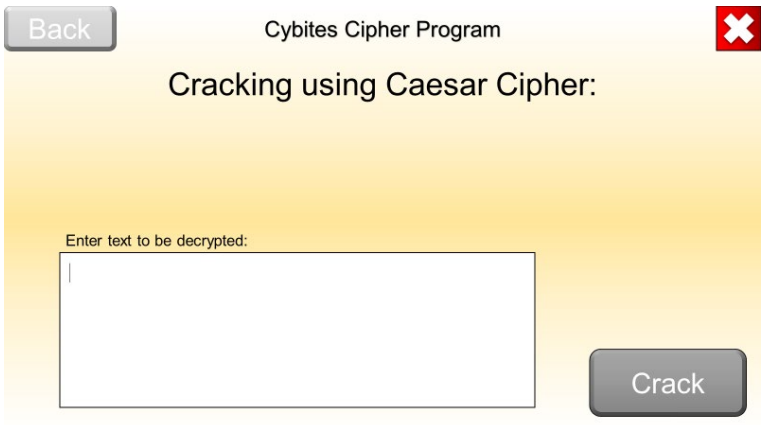


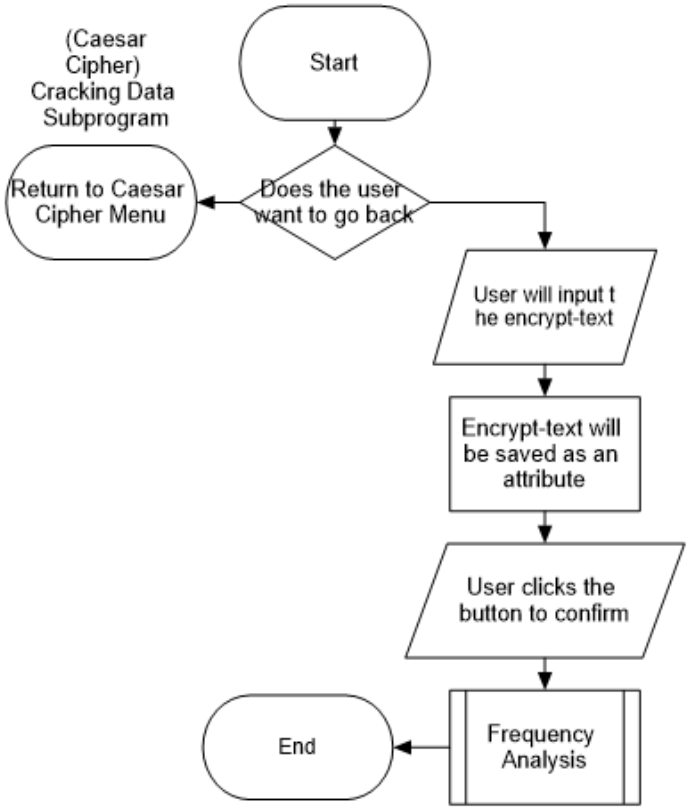


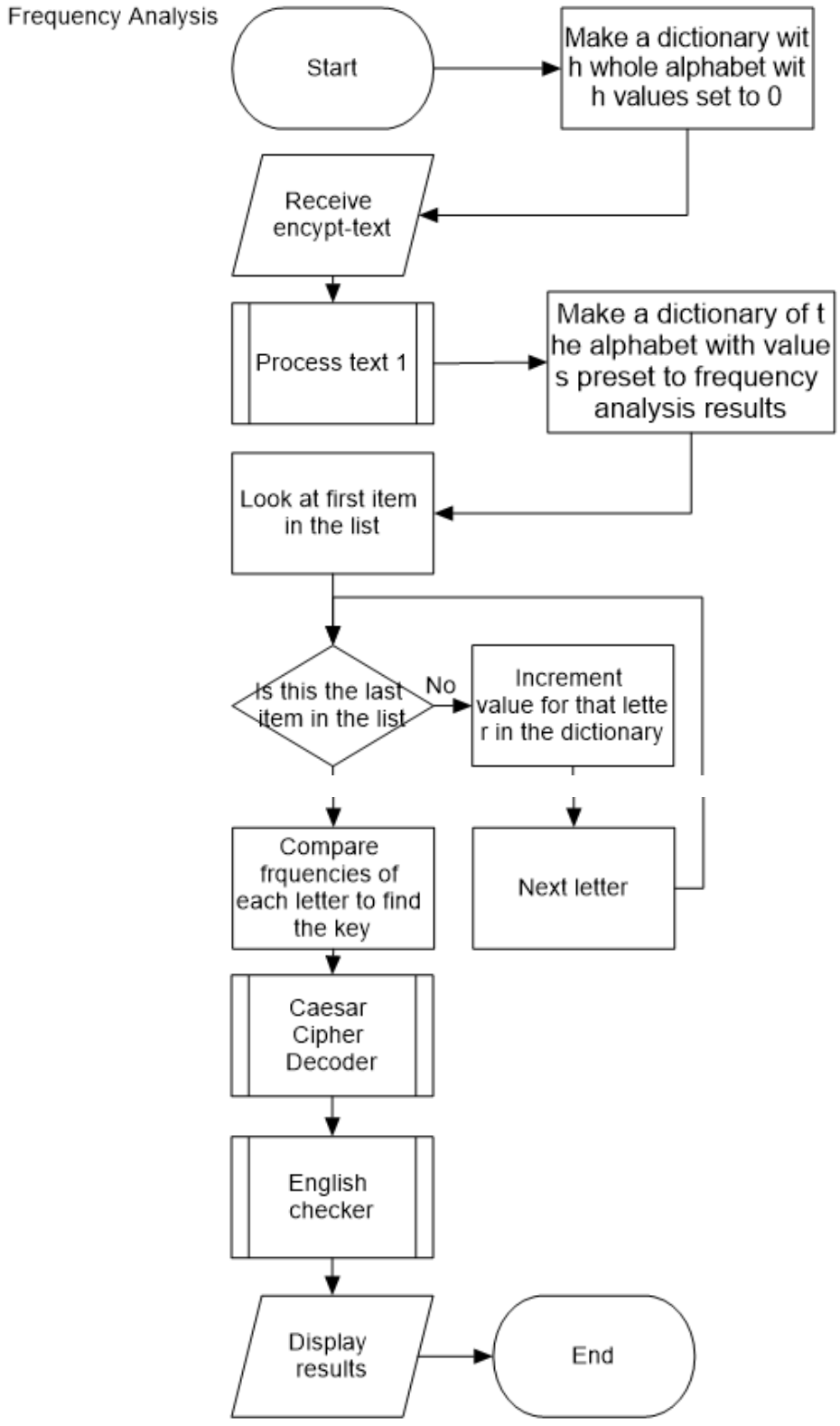
Caesar Cipher Cracking

Success Criteria Aim:

6, 7


Back
Cybites Cipher Program
Cracking using Caesar Cipher:
Enter text to be decrypted:
Crack


(Caesar Cipher) Cracking Data Subprogram
Start
Does the user want to go back
Return to Caesar Cipher Menu
User will input t he encrypt-text
Encrypt-text will be saved as an attribute
User clicks the button to confirm
Frequency Analysis
End

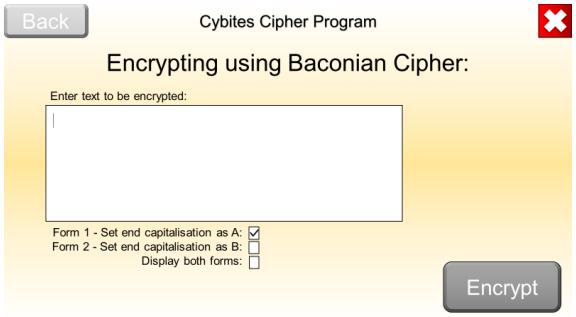


The 'Process Text 1' subprogram will make the string that has been parsed through the main Caesar cipher cracking algorithm a list of characters.

Baconian Cipher Encryption

Success Criteria Aim:

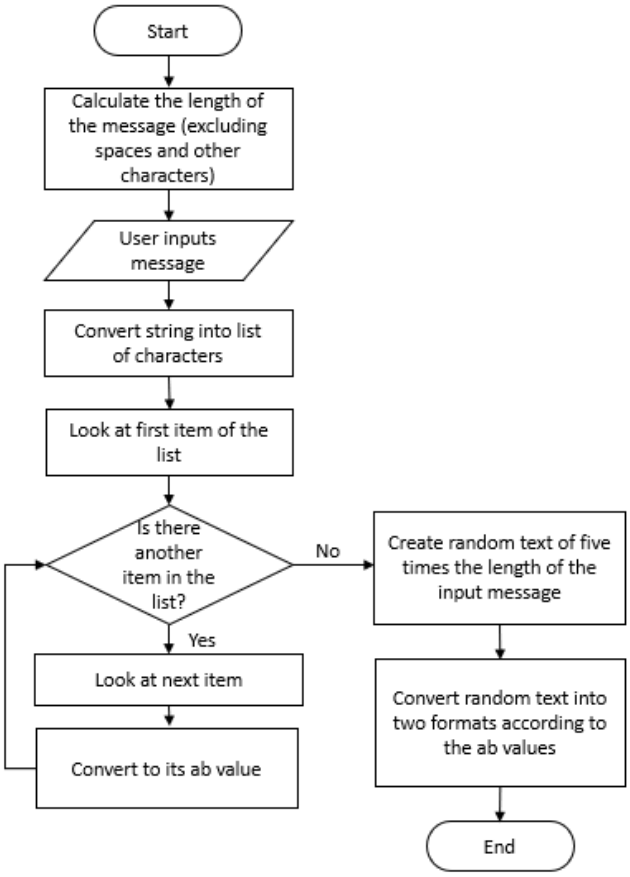
8



(Assuming B is will be the capitalization)

Standard English Walkthrough

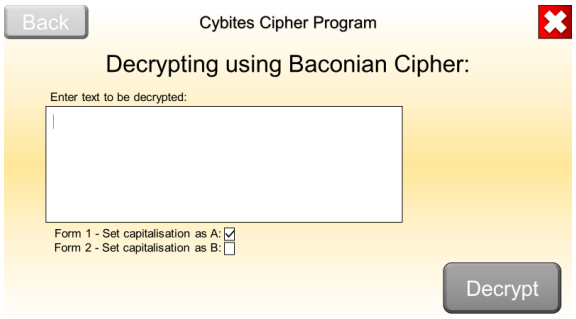
1) Look at first character in the string, find its corresponding AB value (a=aaaaa, b=aaaab ,…,) in list provided and add this to a new list.
2) Split the new list so every A or B value is now separate element in the list.
3) Count how may elements are in the list. Should be a multiple of 5.
4) Create a sentence with all lowercase letters that has the same number of letters corresponding to the number counted in step 3.
5) For every letter in the sentence, if the corresponding position in the list returns the letter A, then leave the letter in the sentence alone, if the corresponding position in the list returns the letter B then make the letter in the sentence the capital version of itself.
6) Display the sentence to the user.

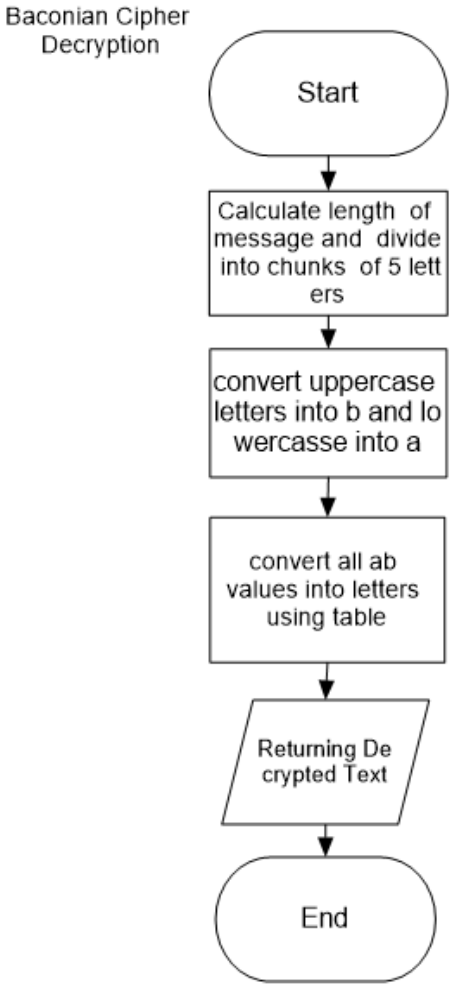


Baconian Cipher Decryption

Success Criteria Aim:

9





Vigenère Cipher Encryption

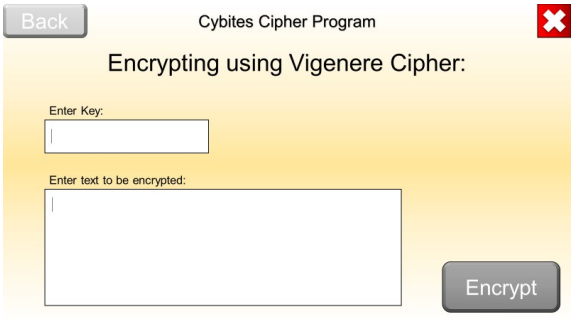
Success Criteria Aim:

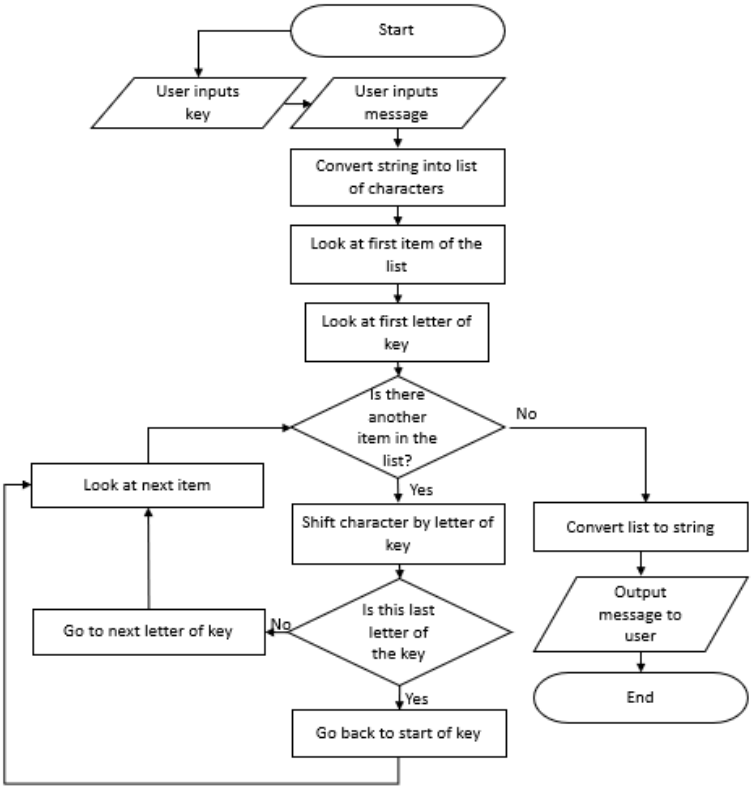
10

Standard English Walkthrough

This subprogram will encrypt the given text by an alphabetical key.

1) Key is converted into a list of its alphabetical equivalents starting from 0 (a=0, b=1, …, z=25)
2) Every alphabetical character in the plain text is converted into a list of its ASCII ordinal equivalents.
3) For the first element in the list of ascii equivalents, the first value of the key list is added and then MOD by 26 to ensure it is still within the range of alphabetical characters and this is added to a new list. This continues throughout the whole of the ascii equivalents list and the key will look every time that it gets to the last letter of the key.
4) Every value in the new list is converted into its alphabetical equivalent and turned into a string.
5) Display string to the user.



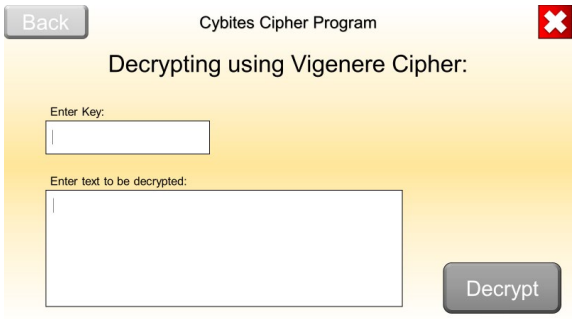


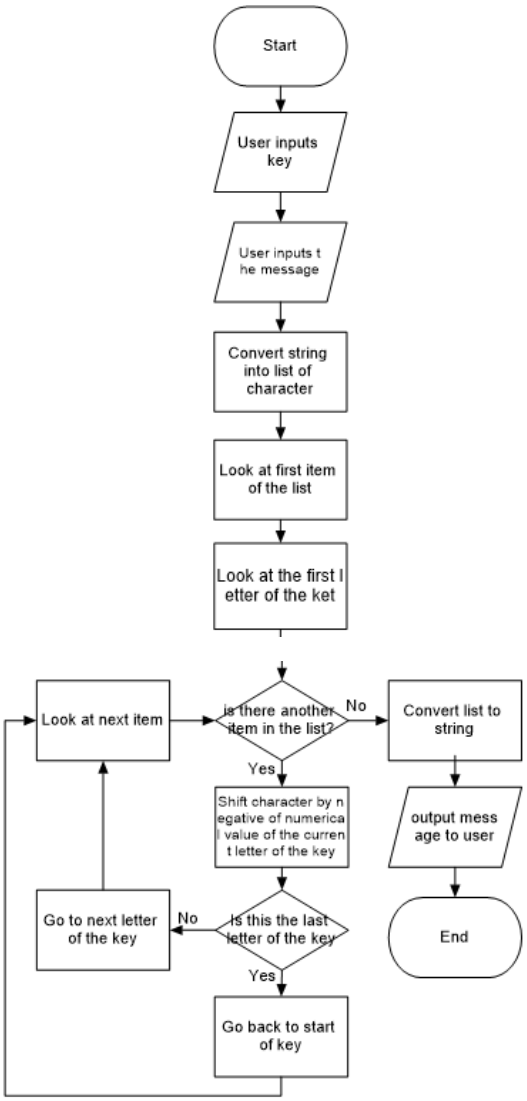

```
key=input(int("Enter key: "))
plaintext=input("Enter text to be encrypted: ")
public function VigenereCipherEncrypt(key, plaintext)
        textlist = convert plaintext to list of characters
        encryptlist = []
        keylist = convert key to list of characters
        for keynum in keylist
                keylist[keynum.position] = ord(keynum)
        position = 0
        for i = 0 to (textlist.length -1)
                if list[i] == "abcdefghijklmnopqrstuvwxyz" then    //If equal to any one of these characters
                        a=convert value to ordinal values
                        a=a+key[position]
                        a=convert back to alphabetical values
                        a.amend to encryptlist
                        position = (position + 1)MOD keylist.length
                next i
        encrypttext= convert encryptlist to a string
        return encrypttext
end function
```

Vigenère Cipher Decryption

Success Criteria Aim:

11



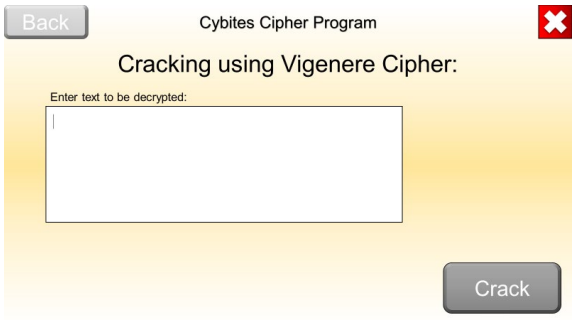


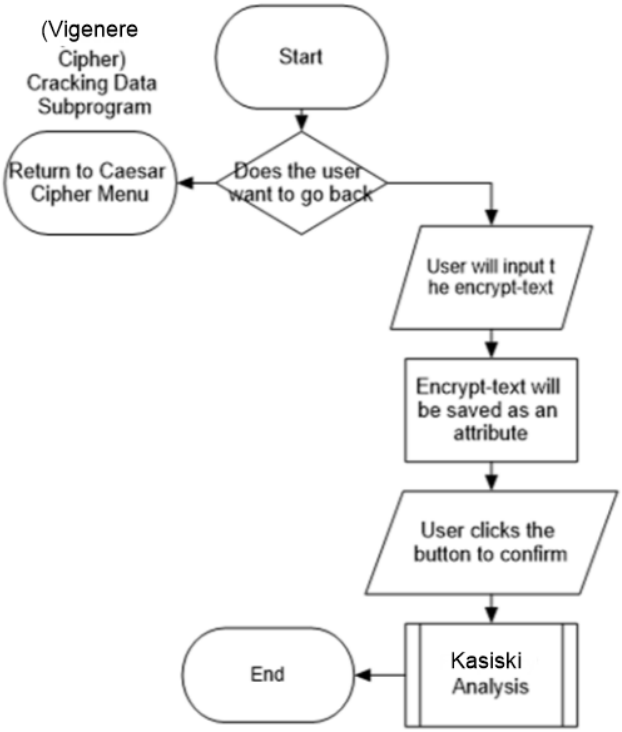
```
key=input(int("Enter key: "))
plaintext=input("Enter text to be encrypted: ")
public function VigenereCipherEncrypt(key, plaintext)
        textlist = convert plaintext to list of characters
        encryptlist = []
        keylist = convert key to list of characters
        for keynum in keylist
                keylist[keynum.position] = ord(keynum)
        position = 0
        for i = 0 to (textlist.length -1)
                if list[i] == "abcdefghijklmnopqrstuvwxyz" then    //If equal to any one of these characters
                        a=convert value to ordinal values
                        a=a-key[position]
                        a=convert back to alphabetical values
                        a.amend to encryptlist
                        position = (position + 1)MOD keylist.length
                next i
        encrypttext= convert encryptlist to a string
        return encrypttext
end function
```

Vigenère Cipher Cracking

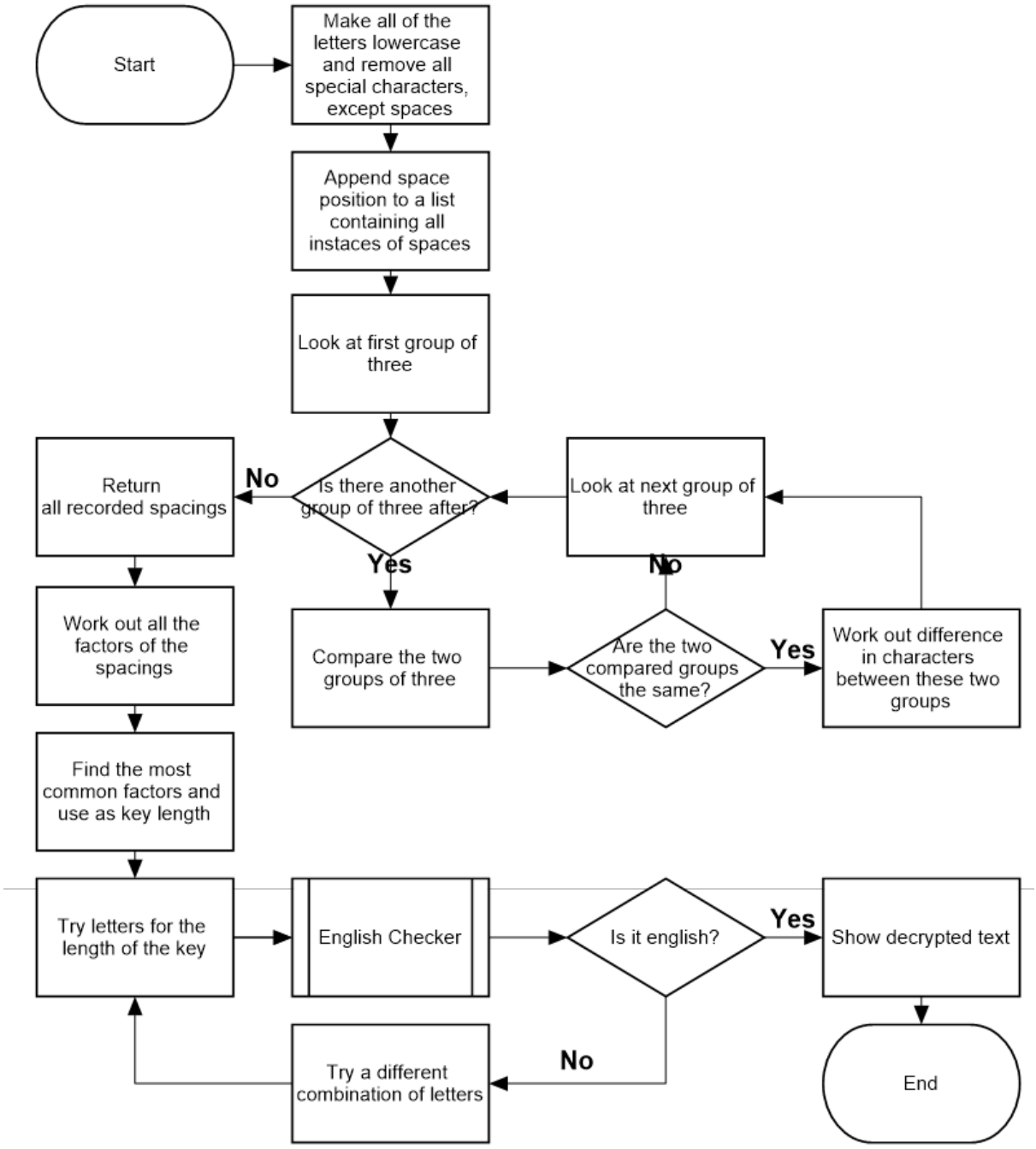
Success Criteria Aim:

12




(Vigenere Cipher) Cracking Data Subprogram
Start
Does the user want to go back
Return to Caesar Cipher Menu
User will input t he encrypt-text
Encrypt-text will be saved as an attribute
User clicks the button to confirm
Kasiski Analysis
End

Kasiski Analysis



Data and Validation

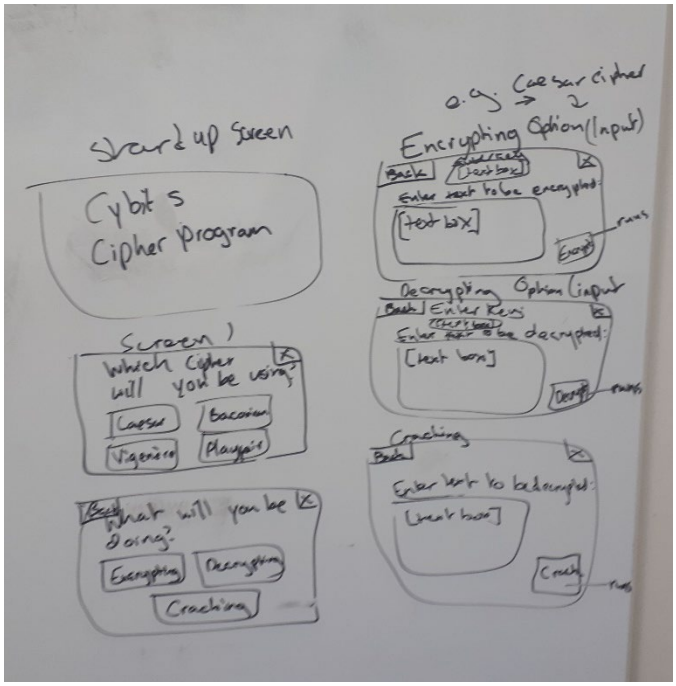


Figure 31 Initial ideas sketch

Above is a rough sketch that was drawn to demonstrate the basic input screens using Caesar Cipher as the chosen cipher. There are fairly few inputs required from the user in this program but each one is crucial to the success of the programs purpose.

All possible inputs for the program:

- Click button to go to selected cipher.
  - Validation: Does the cipher go to the right cipher subprogram menu?
- Click button to go to selected process.
  - Validation: Does the cipher go to the right process subprogram for the right cipher?
- Type text to give key (depends on process chosen).
  - Validation: For Caesar Cipher – The key must be a numerical value.
  - Validation: For Baconian Cipher – Two tick boxes are provided, and user must only be able to select one of the two options.
  - Validation: For Vigenère Cipher – The key must be a string of alphabetical values.
- Type text to give text to be changed by previously selected process.
  - Validation: Text must be in the format of a string.
  - Clarification: Might want to alert the user as to no special characters or numbers will be shifted or may be removed in some ciphers.
- Click button to go back.
- Click button to cancel the processing of the text.
- Click button to exit.
- Click button to copy output text into clipboard.

Test Data

For any ***TEST DATA*** (long pieces of Test Data) see **Appendix**.

Test Number	**Success Criteria**	**Test Data**	**Expected Result**	**Actual Result**	**Success Criteria Met**
1	1	N/A	GUI running in separate window.		
2	2	N/A	Menu with different Cipher buttons.		
3	3	Caesar Selected	Menu with Encrypt, Decrypt, Crack buttons.		
4	3	Baconian Selected	Menu with Encrypt, Decrypt buttons.		
5	3	Vigenère Selected	Menu with Encrypt, Decrypt, Crack buttons.		
6	4, 14	Caesar Selected/ 9/ ***TEST DATA 1***	Result of encryption displayed in the GUI. ***TEST DATA 2***		

7	5, 14	Caesar Selected/ 9/ ***TEST DATA 2***	Result of decryption displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		
8	7, 13i, 14	Caesar Selected/ ***TEST DATA 2***	Result of cracking displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		
9	8	Baconian Selected/ "shantaram"	Result of encryption displayed in the GUI. Random letters with following format: baaab aabbb aaaaa abbaa baaba aaaaa baaaa aaaaa ababb b = uppercase a = lowercase		
10	9	Baconian Selected/ Random letters with following format: baaab aabbb aaaaa abbaa baaba aaaaa baaaa aaaaa ababb	Result of decryption displayed in the GUI. "shantaram"		

11	10, 14	Vigenère Selected/ sha/ ***TEST DATA 1***	Result of encryption displayed in the GUI. ***TEST DATA 3***		
12	11, 14	Vigenère Selected/ sha/ ***TEST DATA 3***	Result of decryption displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		
13	12, 13i, 14	Vigenere Selected/ ***TEST DATA 3***	Result of cracking displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		
14	13ii, 15	Caesar Selected/ ***ERRONEOUS TEST DATA***	Output of null, not found or False.		
15	13ii, 15	Vignere Selected/ ***ERRONEOUS TEST DATA***	Output of null, not found or False.		
16	16i	N/A	Reset button		
17	16ii	N/A	Switch Cipher buttons		

18	16iii	N/A	Exit button		
19	17	N/A	Copy button or selected text with right click menu. Screenshot of data outside the program (e.g. notepad)		

Data Dictionary

Data Item	Data Type	Description	Example	Validation
CaesarCipherKey	Integer	Key used to encrypt/ decrypt	4	Checks that key is integer
CaesarCipherText	String	Data to encrypt/ decrypt	“Hello World!”	N/A
BaconianCipherText	String	Data to encrypt/ decrypt	“Hello World”	N/A
VigenereCipherKey	Single String with only letters	Key used to encrypt/ decrypt	“Spam”	Checks that the string consists of only alphabetical characters and no spaces or special characters.
VigenereCipherText	String	Data to encrypt/ decrypt	“Hello World!”	N/A

Implementation

Caesar Cipher

Caesar Cipher Version 1.0

First, I worked on encryption as a standalone program as I was aware that to create a decryption program it would only require minor changes to the shift.

Brief walkthrough of program:

1. Runs CaesarCipher().
2. Runs CCData().
3. Takes an integer input for the key from the terminal.
4. Takes an input for the text to be encrypted from the terminal.
5. Converts input for the text into a list.
6. Runs CCEncrypt(textlist, key).
7. Creates an empty list called encryptlist.
8. Runs a for loop through textlist with each variable in the loop named ‘a’.
    a. Checks if a is a letter
    b. Creates a new variable with the value that is brought back from convert(a, key)
    c. Adds value to encryptlist
9. Joins encrypt list as a string and prints it to the terminal to the user.

The primary aim of this first version was to complete a Caesar cipher shift that works regardless of any formatting errors.

Success Criteria Aim:

4iii

```
##the primary aim of this version is to complete a caesar cipher shift tha
t will work even if there are formatting errors (Complete)

def CCData():
    global key
    key= int(input("Please enter key: "))
    plaintext= input("Please enter the text to be encrypted: ")
    global textlist
    templist=list(plaintext)
    textlist=templist
    return textlist

def CCEncrypt(letterlist, shift):
    encryptlist=[]
    for a in textlist:
        if a.isalpha()==True:   ##Checks that letter is in the alphabet
        # if ord(a)<=ord("z"):            ##This  code  has  been  commented
out:
```

```
        #     if ord(a)>=ord("a"):  ##The function .isalpha() replaced this
            b=convert(a,shift)
            encryptlist.append(b)
        else:
            encryptlist.append(a)
    joined="".join(map(str,encryptlist))
    print ("The following is the encrypted text:\n"+joined)

def convert(letter, shift):
    l=chr(((ord(letter)-96)+shift)%26+96)
    return l

def CaesarCipher():
    CCData()
    CCEncrypt(textlist,key)

CaesarCipher()
```

The `CCData()` function requests several inputs from the user with the current interface for the program being the python terminal. It then converts the user's string input into a list of all the characters in the string including any numbers or special characters.

The `CCEncrypt(letterlist,shift)` function goes through the list of letters that have been parsed through the parameters and creates a new list of characters that have been shifted by the key value parsed through the shift parameter before making the list into a string of characters and printing it to the terminal.

The `convert(letter, shift)` function returns `l` which is a variable containing the shifted version of the letter that has been passed through the parameters.

The `CaesarCipher()` procedure calls the functions `CCData()` and `CCEncrypt(textlist,key)` so that when run, only this function needs to be called in order to call both of the other functions listed.

Problems with Version 1.0:

- Global variables are set within a function
- All user interaction is within the terminal
- No validation of data types

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be encrypted.	Standard Data The test data is input through the terminal.	Key = 5 Text = the	Encrypted Text: ymj	Please enter key: 5 Please enter the text to be encrypted: the The following is the encrypted text: ymj	Pass
2	Checking that one lowercase sentence can be encrypted. (Checks spaces are unchanged)	Standard Data -	Key = 5 Text = the quick brown fox jumped over the lazy dog	Encrypted Text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Please enter key: 5 Please enter the text to be encrypted: the quick brown fox jumped over the lazy dog The following is the encrypted text: ymj v`nhp gwtbs ktc o`ruji tajw ymj qfed itl	Fail or Pass (with error on z = `)
3	Checking that one mixed case sentence can be encrypted.	Standard Data -	Key = 5 Text = The QUick brown FOX jumped over the lazy doG	Encrypted Text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Please enter key: 5 Please enter the text to be encrypted: The QUick brown FOX jumped over the lazy doG The following is the encrypted text: smj ptnhp gwtbs enw o`ruji tajw ymj qfed itf	Fail
4	Checking that one uppercase	Standard Data -	Key = 5	Encrypted Text: ymj vznhp gwtbs ktc ozruji	Please enter key: 5 Please enter the text to be encrypted: THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG The following is the encrypted text: sgd pthbj aqnvm enw itlodc nudq sgd k`yx cnf	Fail

	sentence can be encrypted.		Text = THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG	tajw ymj qfed itl		
5	Checking that numbers and special characters are not affected	Standard Data -	Key: 5 Text: the & the 1	ymj & ymj 1	Please enter key: 5 Please enter the text to be encrypted: the & the 1 The following is the encrypted text: ymj & ymj 1	Pass

Table 1 Testing Caesar Cipher V1.0

Version 1.0 Review

Success Criteria Met:

N/A

- Only works for lowercase characters
- Test Number 2's error of the special character is remedied in Caesar Cipher Version 1.4.
- Test Number 3 and 4's error of not being able to shift uppercase letters properly is remedied in Caesar Cipher Version 1.1.

Caesar Cipher Version 1.1

Within the next version, the only changes that were made were within the CCData() function.

The primary aim of this version was to allow capital letters to be shifted correctly. I did this by making all capital letters into a lowercase letter. This was successfully completed.

Success Criteria Met:

4iii

```
def CCData():
    global key
    key= int(input("Please enter key: ")) #no validation for the key, assu
med to be an integer
    plaintext= input("Please enter the text to be encrypted: ")
    global textlist
    templist=list(plaintext)
    ##making all capital letters lowercase
    textlist=[]
    lowerA=ord("a")
    capitalA=ord("A")
    capToLowConstant=lowerA-capitalA
    print(capToLowConstant)
    for i in templist:
        if i.isalpha()==True:
            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
                j=chr((ord(i)+capToLowConstant))
                textlist.append(j)
            else:
                 textlist.append(i)
        else:
            textlist.append(i)
    print(textlist)
    return textlist
```

The CCData() function now converts every capital letter, that is encountered while going through the text list given, to a lowercase letter.

Problems with Version 1.1:

- Global variables are set within a function
- All user interaction is within the terminal
- No validation of data types
- Textlist is returned within the CaesarCipher() function, but this is not actually used.

Caesar Cipher Version 1.2

The only changes made to the last version to produce this version were in CCData(), but also the CaesarCipher() function as previously stated was changed to become CaesarCipherEncryption().

The primary aim of this version was to add validation to the key, otherwise the program would encounter an error if it encounters any other data type other than an integer. This was successfully accomplished.

Success Criteria Aim:

4ii, 4iii

```
def CCData():
    global key
    key= input("Please enter key: ")         ##validation

    while key.isdigit()==False:
        print("Please enter an integer value")
        key= input("Please enter key: ")
    key=int(key)
    plaintext= input("Please enter the text to be encrypted: ")
    global textlist
    templist=list(plaintext)
    textlist=[]
    lowerA=ord("a")
    capitalA=ord("A")
    capToLowConstant=lowerA-capitalA
    print(capToLowConstant)
    for i in templist:
        if i.isalpha()==True:
            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
                j=chr((ord(i)+capToLowConstant))
                textlist.append(j)
            else:
                 textlist.append(i)
        else:
            textlist.append(i)
    print(textlist)
    return textlist
...

def CaesarCipherEncryption():
    CCData()
    CCEncrypt(textlist,key)

CaesarCipherEncryption()
```

The `CCData()` function now will take an input for the key, but then immediately check it to see if it is an integer and if any other data type is given then the user will be asked to input a key of the right data type and this will loop until the terminal is closed or the user gives a correct input. It then makes this key variable an integer value.

The `CaesarCipherEncryption()` function is the same function as CaesarCipher(), but the name was changed to avoid collisions when the decryption function is created.

Problems with Version 1.2:

- Global variables are set within a function
- All user interaction is within the terminal
- Textlist is returned within the CaesarCipherEncryption() function, but this is not actually used.

Caesar Cipher Version 1.3

A separate function was made as when decomposed the program is required to convert letters to lowercase later in the program (e.g. within in the Vigenère cipher). The rest of the program remains the same.

The primary aim of this version was the extract the relevant code from CCData() and make it reusable.

Success Criteria Aim:

4ii, 4iii

```
def CCData():
    global key
    key = "letter"       ##validation
    while key.isdigit()==False:
        print("Please enter an integer value")
        key= input("Please enter key: ")
    key=int(key)
    plaintext= input("Please enter the text to be encrypted: ")
    global textlist
    templist=list(plaintext)
    ##making all capital letters lowercase
    textlist=processtext(templist)
    print(textlist)


#Have made a process text function so that this can be reused in the vigen
ere cipher
def processtext(templist):
    textlist=[]
    lowerA=ord("a")
    capitalA=ord("A")
```

```
    capToLowConstant=lowerA-capitalA
    for i in templist:
        if i.isalpha()==True:
            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
                j=chr((ord(i)+capToLowConstant))
                textlist.append(j)
            else:
                 textlist.append(i)
        else:
            textlist.append(i)
    return textlist
```

The CCData() function had a lot of code removed and put into a separate function called processtext(templist).

The processtext(templist) function just converts all uppercase letters into lowercase letters as previously demonstrated.

Problems with Version 1.3:

- Global variables are set within a function
- All user interaction is within the terminal
- Textlist is returned within the CaesarCipherEncryption() function, but this is not actually used.
- An error was found that when it converted any letter to the letter 'z' it was actually returning the special character '`'.

Caesar Cipher Version 1.4

The only changes made to the program were in the convert() function.

The primary aim of this version was to find out how to stop the program returning ` instead of z which was caused by the fact 26MOD26 = 0 as 26 is the number corresponding to z in the alphabet.

Success Criteria Aim:

4ii, 4iii

```
def convert(letter, shift):
    l=chr(((ord(letter)-96)+shift)%26+96)
    if l =="`":          ##added code to counter the wrong character
        l="z"
    return l
```

The convert(letter, shift) function had an if statement added to it so that if there was a ` produced, this character would get replaced with a 'z'.

Problems with Version 1.4:

- Global variables are set within a function
- All user interaction is within the terminal
- Textlist is returned within the CaesarCipherEncryption() function, but this is not actually used.

Caesar Cipher Version 1.5

In this version, the previously written processtext() function is made redundant and replaced with a more astute one line of code, which was a predefined function, available in pythons documentation.

The aim of this version was to remove the global variables within the CCData() function. Successfully completed.

Success Criteria Aim:

4ii, 4iii

```
def CCData():
    key = "letter"       ##making it false so the while loop will run
    while key.isdigit()==False:
        print("Please enter an integer value")
        key= input("Please enter key: ")
    key=int(key)
    plaintext= input("Please enter the text to be encrypted: ")
    templist=list(plaintext)
    ##making all capital letters lowercase
    textlist = [m.lower() for m in templist]  ##replaces processtext()
    return textlist, key

##Function has been made redundant
##def processtext(templist):
##    textlist=[]
##    lowerA=ord("a")
##    capitalA=ord("A")
##    capToLowConstant=lowerA-capitalA
##    for i in templist:
##        if i.isalpha()==True:
##            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
##                j=chr((ord(i)+capToLowConstant)) ##
##                textlist.append(j)
##            else:
##                 textlist.append(i)
##        else:
##            textlist.append(i)
##    #print (textlist)     #checks that function was as original
##    return textlist
```

```
def CCEncrypt(letterlist, shift):
    encryptlist=[]
    for a in letterlist:
        if a.isalpha()==True:   ##verifies that letter is in the alphabet
            b=convert(a,shift)
            encryptlist.append(b)
        else:
            encryptlist.append(a)
    joined="".join(map(str,encryptlist))
    print ("The following is the encrypted text:\n"+joined)

def convert(letter, shift):
    l=chr(((ord(letter)-96)+shift)%26+96)
    if l =="`":
        l="z"
    return l

def CaesarCipherEncryption():
    print("Caesar Cipher > Encryption")
    textlist, key = CCData()    ##using this code, it gets rid of the glob
al variables
    CCEncrypt(textlist,key)

CaesarCipherEncryption()
```

Problems with Version 1.5:

- All user interaction is within the terminal

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be encrypted.	Standard Data The test data is input through the terminal.	Key = 5 Text = the	Encrypted Text: ymj	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: the The following is the encrypted text: ymj	Pass
2	Checking that one lowercase sentence can be encrypted. (Checks spaces are unchanged)	Standard Data -	Key = 5 Text = the quick brown fox jumped over the lazy dog	Encrypted Text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: the quick brown fox jumped over the lazy dog The following is the encrypted text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Pass
3	Checking that one mixed case sentence can be encrypted.	Standard Data -	Key = 5 Text = The QUick brown FOX jumped over the lazy doG	Encrypted Text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: The QUick brown FOX jumped over the lazy doG The following is the encrypted text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Pass
4	Checking that one uppercase sentence can be encrypted.	Standard Data -	Key = 5 Text = THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG	Encrypted Text: ymj vznhp gwtbs ktc ozruji	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG The following is the encrypted text: ymj vznhp gwtbs ktc ozruji tajw ymj qfed itl	Pass

				tajw ymj qfed itl		
5	Checking that numbers and special characters are not affected	Standard Data -	Key: 5 Text: the & the 1	ymj & ymj 1	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: the & the 1 The following is the encrypted text: ymj & ymj 1	Pass

Table 2 Testing Caesar Cipher V1.0

Version 1.5 Review

Success Criteria Met:

4ii, 4iii

All tests for this version have been passed therefore it should meet the stakeholder requirements. .

Caesar Cipher Version 2.0

Next, I focused on making the decryption program which was a change in the direction of the shift.

The aim of this program was to alter the previous encryption program in order to create a decryption program. This was originally attempted at earlier versions, but shown below is the version that corresponds to Version 1.5, but with the reverse shift. This was successfully completed.

Success Criteria Aim:

5ii, 5iii

```
##Gives results where all alphabetical characters are lowercase

def CCData():
    key = "letter"
    while key.isdigit()==False:
        print("Please enter an integer value")
        key= input("Please enter key: ")
    key=int(key)
    plaintext= input("Please enter the text to be decrypted: ")
    templist=list(plaintext)
    textlist = [m.lower() for m in templist]
    return textlist, key

def CCDecrypt(letterlist, shift):
    decryptlist=[]
    for a in letterlist:
        if a.isalpha()==True:
            b=convert(a,shift)
            decryptlist.append(b)
        else:
            decryptlist.append(a)
    joined="".join(map(str,decryptlist))
    print ("The following is the decrypted text:\n"+joined)

def convert(letter, shift):
    l=chr(((ord(letter)-96)-shift)%26+96)    # This becomes a minus shift
    if l =="`":
        l="z"
    return l

def CaesarCipherDecryption():
    print("Caesar Cipher > Decryption")
    textlist, key = CCData()
    CCDecrypt(textlist,key)
```

`CaesarCipherDecryption()`

The `convert(letter, shift)` function has had the addition of the shift changed to a subtraction so each letter will shift backwards by the amount given.

Problems with Version 2.0

- All user interaction is within the terminal
- The encryption and decryption programs have minor differences so they ideally should be within one program to avoid code redundancy.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be decrypted.	Standard Data The test data is input through the terminal.	Key = 5 Text = the	Decrypted Text: ocz	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be decrypted: the The following is the decrypted text: ocz	Pass
2	Checking that one lowercase sentence can be decrypted. (Checks spaces are unchanged)	Standard Data -	Key = 5 Text = the quick brown fox jumped over the lazy dog	Decrypted Text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be decrypted: the quick brown fox jumped over the lazy dog The following is the decrypted text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Pass
P3	Checking that one mixed case sentence can be decrypted.	Standard Data -	Key = 5 Text = The QUick brown FOX jumped over the lazy doG	Decrypted Text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be decrypted: The QUick brown FOX jumped over the lazy doG The following is the decrypted text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Pass

4	Checking that one uppercase sentence can be decrypted.	Standard Data -	Key = 5 Text = THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG	Decrypted Text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be decrypted: THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG The following is the decrypted text: ocz lpdxf wmjri ajs ephkzy jqzm ocz gvut yjb	Pass
5	Checking that numbers and special characters are not affected	Standard Data -	Key: 5 Text: the & the 1	Decrypted Text: ocz & ocz 1	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be decrypted: the & the 1 The following is the decrypted text: ocz & ocz 1	Pass

Table 3 Testing Caesar Cipher V2.0

Version 2.0 Review

Success Criteria Met:

5ii, 5iii

All tests for this version have been passed therefore it should meet the stakeholder requirements. .

Caesar Cipher Version 3.0

After I had a program for both encryption and decryption, I wanted to put it into one program and this required me to make another variable called mode, which determines whether the shift will be positive or negative.

The aim of this version was to combine encryption and decryption to reduce redundancy of code within the Caesar Cipher. This was completed successfully.

Success Criteria Aim:

4ii, 4iii, 5ii, 5iii

```
#Gives results where all alphabetical characters are lowercase

def CCData():
    key = "letter"
    while key.isdigit()==False:
        print("Please enter an integer value")
        key= input("Please enter key: ")
    key=int(key)
    plaintext= input("Please enter the text to be encrypted: ")
    templist=list(plaintext)
    textlist = [m.lower() for m in templist]
    return textlist, key

def CCChange(letterlist, shift, rightorleft):
    encryptlist=[]
    for a in letterlist:
        if a.isalpha()==True:
            b=convert(a,shift, rightorleft)
            encryptlist.append(b)
        else:
            encryptlist.append(a)
    joined="".join(map(str,encryptlist))
    print ("The following is the encrypted text:\n"+joined)

    ##mode + or -
def convert(letter, shift, rightorleft):
    l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
    if l =="`":
        l="z"
    return l

def CaesarCipherDecryption():
    mode = -1
    print("Caesar Cipher > Decryption")
    textlist, key = CCData()
    CCChange(textlist,key,mode)
```

```
def CaesarCipherEncryption():
    mode = +1
    print("Caesar Cipher > Encryption")
    textlist, key = CCData()
    CCChange(textlist,key,mode)

##CaesarCipherEncryption()
##CaesarCipherDecryption()
```

The `convert(letter, shift, rightorleft)` function has another parameter that will pass the variable from either `CaesarCipherEncryption()` as +1 or `CaesarCipherDecryption()` as -1, to be multiplied by the shift's magnitude and hence determines the direction of the shift.

The `CaesarCipherDecryption()` function has the variable mode added as -1.

The `CaesarCipherDecryption()` function has the variable mode added as +1.

Problems with Version 3.0

- All user interaction is within the terminal
- Code should be in an object with encryption and decryption as methods

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be encrypted.	Standard Data The test data is input through the terminal.	Key = 5 Text = the	Encrypted Text: ymj	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: the The following is the encrypted text: ymj	Pass
2	Checking that a sentence containing lowercase, uppercase and special characters can be encrypted.	Standard Data -	Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog	Encrypted Text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl	Caesar Cipher > Encryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: The Quick Brown Fox! jumped over the 1 lazy dog The following is the encrypted text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl	Pass
3	Checking that one lowercase word can be decrypted.	Standard Data -	Key = 5 Text = the	Decrypted Text: ocz	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: the The following is the encrypted text: ocz	Pass

4	Checking that a sentence containing lowercase, uppercase and special characters can be decrypted.	Standard Data -	Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog	Decrypted Text: ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb	Caesar Cipher > Decryption Please enter an integer value Please enter key: 5 Please enter the text to be encrypted: The Quick Brown Fox! jumped over the 1 lazy dog The following is the encrypted text: ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb	Pass

Table 4 Testing Caesar Cipher V3.0

Version 3.0 Review

Success Criteria Met:

4ii, 4iii, 5ii, 5iii

All tests for this version have been passed therefore it should meet the stakeholder requirements. .

Caesar Cipher Version 4.0

I made this version with the intent to put the Caesar Cipher into a class with attributes and methods so that the information entered become attributes of an object that is created.

The primary aim of this version is to make Caesar a class and it was attempted at earlier stages, but the progressions are matched with previous versions. This was successfully completed, although a bound method error was encountered, but was overcome.

Success Criteria Aim:

4ii, 4iii, 4iv, 5ii, 5iii, 5iv

```
class Caesar:
    def __init__(self, ciphertext, shift, process):
        self.ciphertext=ciphertext
        self.shift=shift
        self.process=process

    def CCProcess(self, rightorleft):
        processlist=[]
        for a in self.ciphertext:
            if a.isalpha()==True:
                b=self.convert(a,self.shift, rightorleft)
                processlist.append(b)
            else:
                processlist.append(a)
        joined="".join(map(str,processlist))
        return joined

    def convert(self, letter, shift, rightorleft):
        l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
        if l =="`":
            l="z"
        return l

    def CaesarCipherDecryption(self):
        mode = -1
        print("Caesar Cipher > Decryption")
        print ("The following is the decrypted text:")
        result = self.CCProcess(mode)
        return result


    def CaesarCipherEncryption(self):
        mode = +1
        print("Caesar Cipher > Encryption")
        print ("The following is the encrypted text:")
        result = self.CCProcess(mode)
        return result
```

```
def CCData():
    shift=”a”
    while shift.isdigit()==False:
        print(“Please enter an integer value”)
        shift= input(“Please enter key: “)
    shift=int(shift)
    plaintext= input(“Please enter the text to be en/decrypted: “)
    templist=list(plaintext)
    textlist = [m.lower() for m in templist]
    return Caesar(textlist, shift, True)

test = CCData()
##print(test.CaesarCipherEncryption())
##print(test.CaesarCipherDecryption())
```

The class `Caesar` was created with attributes; ciphertext, shift and process and methods; CCProcess(), convert(), CaesarCipherEncryption(), CaesarCipherDecryption().

The `CCData()` function is separate but the same and the object is initiated through this.

Problems with Version 4.0

- All user interaction is within the terminal
- Validation for the input from the terminal will not transfer out of the terminal when interacting with a GUI

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be encrypted.	Standard Data The test data is input through the terminal.	Key = 5 Text = the	Encrypted Text: ymj	Please enter an integer value Please enter key: 5 Please enter the text to be en/decrypted: the Caesar Cipher > Encryption The following is the encrypted text: ymj	Pass
2	Checking that a sentence containing lowercase, uppercase and special characters can be encrypted.	Standard Data -	Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog	Encrypted Text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl	Please enter an integer value Please enter key: 5 Please enter the text to be en/decrypted: The Qui ck Brown Fox! jumped over the 1 lazy dog Caesar Cipher > Encryption The following is the encrypted text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl	Pass
3	Checking that one lowercase word can be decrypted.	Standard Data -	Key = 5 Text = the	Decrypted Text: ocz	Please enter an integer value Please enter key: 5 Please enter the text to be en/decrypted: the Caesar Cipher > Decryption The following is the decrypted text: ocz	Pass
4	Checking that a sentence containing lowercase, uppercase and special	Standard Data -	Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog	Decrypted Text: ocz lpdxf wmjri ajs!	Please enter an integer value Please enter key: 5 Please enter the text to be en/decrypted: The Qui ck Brown Fox! jumped over the 1 lazy dog Caesar Cipher > Decryption The following is the decrypted text: ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb	Pass

	characters can be decrypted.			ephkzy jqzm ocz 1 gvut yjb		

Table 5 Testing Caesar Cipher V4.0

Version 4.0 Review

Success Criteria Met:

4ii, 4iii, 5ii, 5iii

All tests for this version have been passed therefore it should meet the stakeholder requirements. .

- This version's testing has not been performed in the GUI so 4iv and 5iv cannot be verified.
- The program still encrypts and decrypts as it did before it was put into a class.

Caesar Cipher Version 4.1

At the time of the creation of this version I was working on the GUI so interaction is not purely terminal based, although some print statements remain so I could track what the program is doing when I put in data.

The primary aim of this version was to be able to use it from the GUI, this includes validation. This was successful.

Success Criteria Aim:

4ii, 4iii, 4iv, 5ii, 5iii, 5iv

```
##This version is altered so it works with the GUI
##Redundancy: This version still prints into terminal

class Caesar:
    def __init__(self, ciphertext, shift):
        self.ciphertext=ciphertext
        self.shift=shift

    def CCProcess(self, rightorleft):
        processlist=[]
        for a in self.ciphertext:
            if a.isalpha()==True:
                b=self.convert(a,self.shift, rightorleft)
                processlist.append(b)
            else:
                processlist.append(a)
        joined="".join(map(str,processlist))
        return joined

    def convert(self, letter, shift, rightorleft):
        l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
        if l =="`":
            l="z"
        return l

    def CaesarCipherDecryption(self):
        mode = -1
        print("Caesar Cipher > Decryption")
        print ("The following is the decrypted text:\n")
        result = self.CCProcess(mode)
        return result

    def CaesarCipherEncryption(self):
        mode = +1
        print("Caesar Cipher > Encryption")
        print ("The following is the encrypted text:\n")
        result = self.CCProcess(mode)
```

```
        return result

def CCData(shift,plaintext):        ##validation for key within GUI
    ###while shift.isdigit()==False:
    ###    print("Please enter an integer value")
    ###    shift= input("Please enter key: ")
    ##shift=int(shift)
    templist=list(plaintext)
    textlist = [m.lower() for m in templist]
    return Caesar(textlist, shift)

# caesar = CCData()
# caesar.CaesarCipherDecryption()
```

The `CCData(shift,plaintext)` function has had the inputs and validation removed as these will be input through the GUI. It still converts the text into a list of the lowercase letters from the parameter plaintext.

Problems with Version 4.0

- CCData() function is mostly redundant
- There are print (to terminal) statements that are unnecessary within the code

Caesar Cipher Version 4.2 (Version in use)

This is the current version of the CaesarCipher being used by the GUI. CCData() was removed fully and replaced with the first line in the __init__() function.

The primary aim of this version was to optimise the code for when it is used the GUI.

Success Criteria Aim:

4ii, 4iii, 4iv, 5ii, 5iii, 5iv

```
##This version is altered so it works with the GUI

class Caesar:
    def __init__(self, ciphertext, shift):
        self.ciphertext=list([m.lower() for m in ciphertext])
        ##ciphertext is a list
        self.shift=shift

    def CCProcess(self, rightorleft):
        processlist=[]
        for a in self.ciphertext:
            if a.isalpha()==True:
                b=self.convert(a,self.shift, rightorleft)
                processlist.append(b)
            else:
```

```
                processlist.append(a)
        joined="".join(map(str,processlist))
        return joined

    def convert(self, letter, shift, rightorleft):
        l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
        if l ==`":
            l="z"
        return l

    def CaesarCipherDecryption(self):
        mode = -1
        result = self.CCProcess(mode)
        return result

    def CaesarCipherEncryption(self):
        mode = +1
        result = self.CCProcess(mode)
        return result

#caesar = Caesar("spAm",5)
#print(caesar.CaesarCipherDecryption())
```

Code from GUI about Caesar Cipher within the GUI

This part of the program is mainly initialising the frames for the Caesar cipher

. . . - ***This means that there is more code, but it has been taken out because it doesn't affect the relevant subject version.***

```
import CaesarV5 as CC

class GUI:              ##The GUI is a class so I can switch screens easil
y
    def __init__(self, root):
        self.root = root
        self.main_frame = Frame(root,bg = "Light Yellow")
      ##This frame is constant throughout the program
        self.caesar_frame = Frame(root,bg="Light Yellow")
      ##These have been initialized as frames and contain information in t
      he corresponding functions
        self.main_frame.pack()
      #These are packed so that the program looks more
      orderly and these can be forgotten and reinstated easily
        self.caesar_frame.pack()
         self.caesar_result = Label(self.caesar_frame, background = "Light
      Yel  low")
```

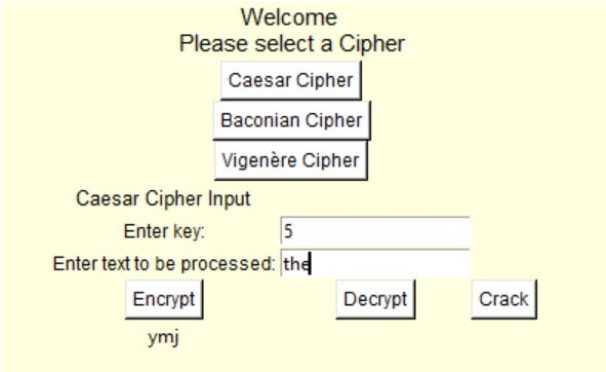
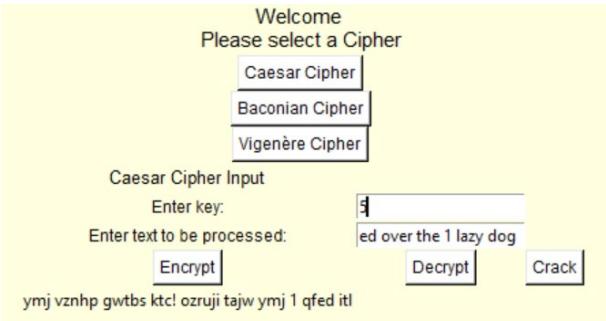
```
        ##These are creating labels to be used later on in the program to di
        sp-lay the result
          self.caesar_result.grid(row=14)
        ##These use grid because within all the frames grid is used, but the
         frames themselves are packed.
              . . .

def main(self):
        welcome=Label(self.main_frame, bg="Light Yellow", font=("Helvetica
        ", 11), text = "Welcome \nPlease select a Cipher").grid(row=0, col
        umn=0)
        firstbutton = Button(self.main_frame, bg="white",font=("Helvetica"
        , 9),text="Caesar Cipher", command=lambda: self.CaesarSelect()).gr
        id(row=2,column=0)
              . . .
        root.mainloop()

    def CaesarSelect(self):
        print("Caesar Cipher Successfully clicked")  ##This will appear on
 the terminal
        self.caesar_frame.pack()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack_forget()
        caesar_frame=self.caesar_frame
        CCnote=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 10
),text="Caesar Cipher Input").grid(row=7, column=0) ##This will appear on
the window
        CCkey=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 9),
text="Enter key:").grid(row=8)
        entry1=Entry(caesar_frame)
        CCtext=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 9)
,text="Enter text to be processed:").grid(row=9)
        entry2 = Entry(caesar_frame)

        entry1.grid(row = 8,column=1)
        entry2.grid(row = 9, column = 1)

        encryptButton = Button(caesar_frame, bg="white",font=("Helvetica",
 9), text = "Encrypt",command=lambda: self.CCEncryption(caesar_frame,entry
1,entry2))
        decryptButton = Button(caesar_frame, bg="white",font=("Helvetica",
 9), text = "Decrypt",command=lambda: self.CCDecryption(caesar_frame,entry
1,entry2))
        crackButton = Button(caesar_frame, bg="white",font=("Helvetica", 9
), text = "Crack", command = lambda: self.CCCrack(caesar_frame,entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid(row = 11, column = 2)
```

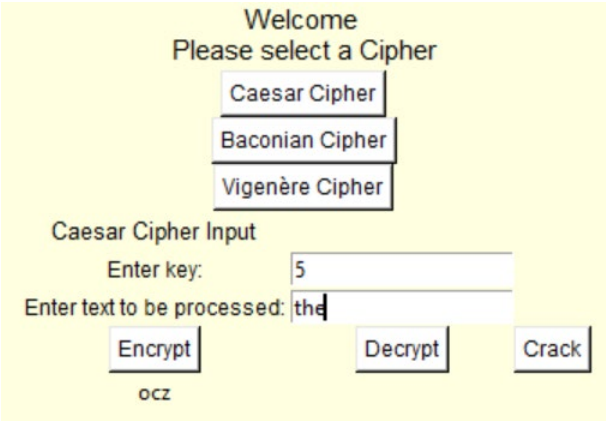
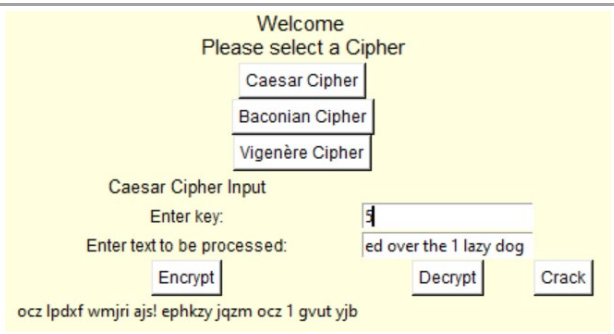
Code from GUI about validation

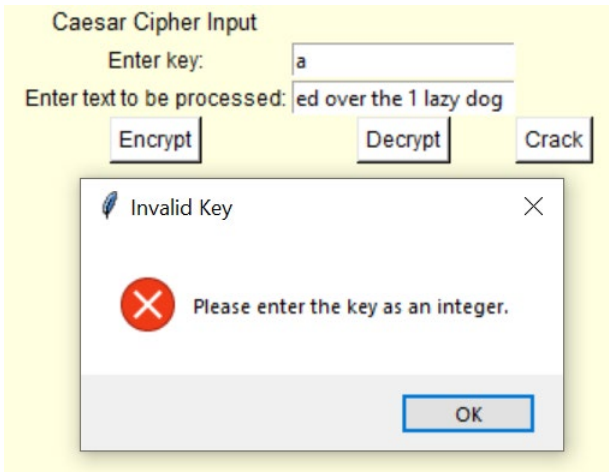
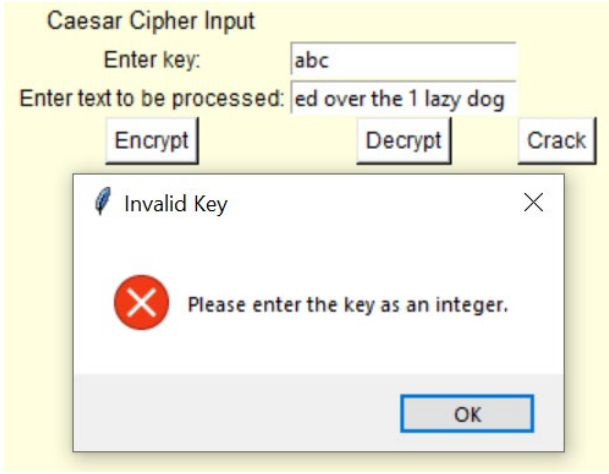
```
    def CCEncryption(self,caesar_frame,entry1,entry2):
        key = entry1.get()
        if key.isdigit() == False:          ##validation
            messagebox.showerror("Invalid Key","Please enter the key as an
integer.")
            entry1.delete(0,"end")
        else:
            key=int(key)
        text = entry2.get()
        showEncrypt="not found"
        caesarE = CC.Caesar(text,key)
        encrypting=caesarE.CaesarCipherEncryption()



        self.caesar_result.config(text=encrypting)
      ##This code changes what is contained in the label in self with what
       has been encrypted

    def CCDecryption(self,caesar_frame,entry1,entry2):
        key = entry1.get()
        text = entry2.get()
        if key.isdigit() == False:          ##validation
            messagebox.showerror("Invalid Key","Please enter the key as an
integer.")
            entry1.delete(10,"end")
        else:
            key=int(key)
        caesarD = CC.Caesar(text,key)
        decrypting=caesarD.CaesarCipherDecryption()
        self.caesar_result.config(text=decrypting)
```

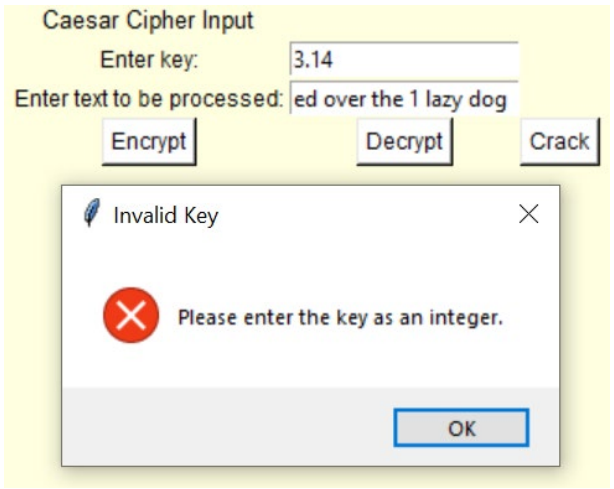
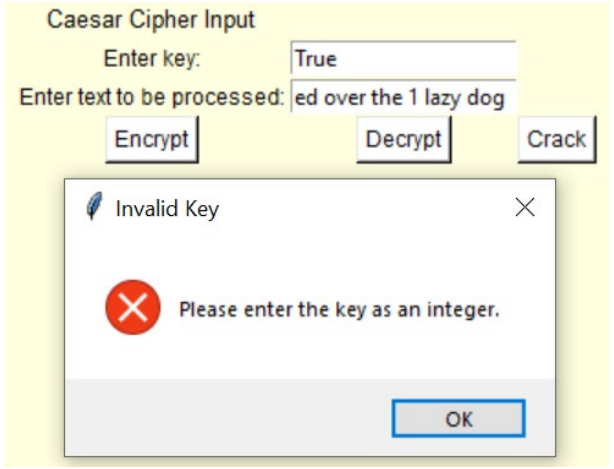
Caesar
Key
Text
CaesarCipherEncryption()
CaesarCipherDecryption()

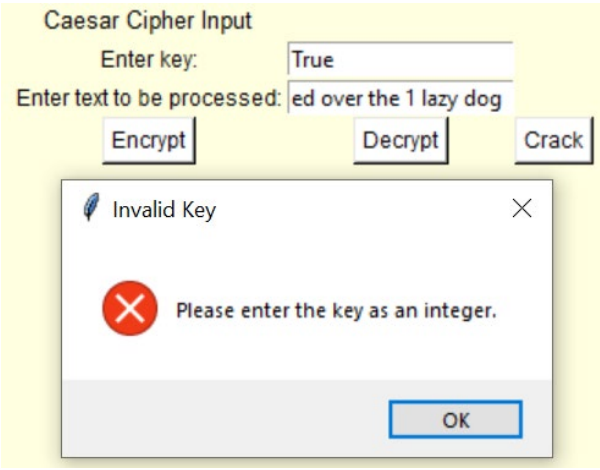
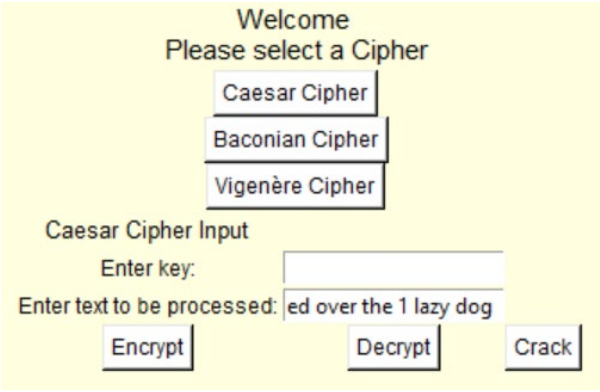
Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that one lowercase word can be encrypted and is returned in the GUI.	Standard Data The test data is input through the GUI.	Caesar Cipher button clicked Key = 5 Text = the Encrypt button clicked	Encrypted Text: ymj		Pass
2	Checking that a sentence containing lowercase, uppercase and special characters can be encrypted and is returned in the GUI.	Standard Data -	Caesar Cipher button clicked Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog Encrypt button clicked	Encrypted Text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl		Pass

3	Checking that one lowercase word can be decrypted and is returned in the GUI.	Standard Data -	Caesar Cipher button clicked Key = 5 Text = the Decrypt button clicked	Decrypted Text: ocz		Pass
4	Checking that a sentence containing lowercase, uppercase and special characters can be decrypted and is returned in the GUI.	Standard Data -	Caesar Cipher button clicked Key = 5 Text = The Quick Brown Fox! jumped over the 1 lazy dog Decrypt button clicked	Decrypted Text: ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb		Pass
5	Checking that a non-integer key creates an error message box for encryption/decryption.	Erroneous Data -	Caesar Cipher button clicked Text = The Quick Brown Fox! jumped over the 1 lazy dog	A message box appears to stop the input being accepted.	Character:	Pass

			Key(s) = Character: a, String: abc, Float: 3.14, Boolean: True Encrypt/Decrypt button clicked		 String:  Float:	

					 Boolean: 

6	Checking that the key input section is cleared after the error box is exited.	Erroneous Data -	Caesar Cipher button clicked Text = The Quick Brown Fox! jumped over the 1 lazy dog Key(s) = abc Encrypt button clicked Message box exited	The key input entry should clear	 OK button clicked: 	Pass

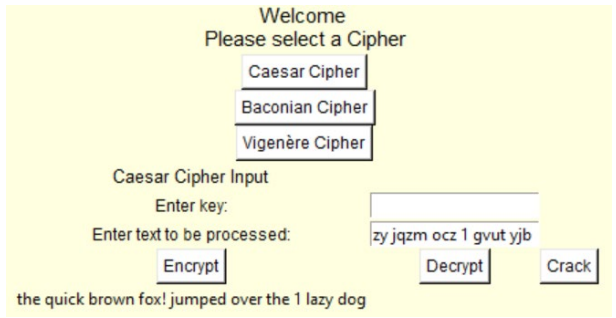
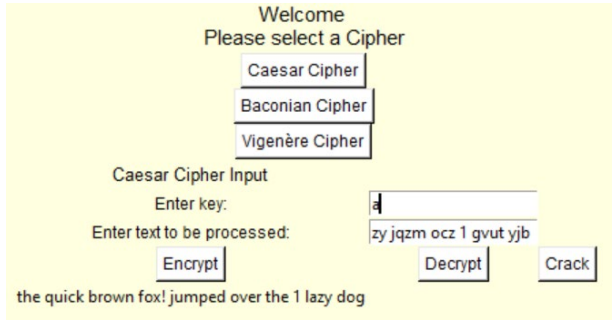
7	Checking that an encrypted sentence can be cracked and returned in the GUI.	Standard Data -	Caesar Cipher button clicked Text = ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb Crack button clicked	Cracked Text: the quick brown fox! jumped over the 1 lazy dog		Pass
8	Checking that an input in the key entry box does not affect the output of the cracked text returned in the GUI.	Standard Data	Caesar Cipher button clicked Key = a Text = ocz lpdxf wmjri ajs! ephkzy jqzm ocz 1 gvut yjb Crack button clicked	Cracked Text: the quick brown fox! jumped over the 1 lazy dog		Pass

Table 6 Testing Caesar Cipher 4.2

Version 3.0 Review

Success Criteria Met:

4ii, 4iii, 4iv, 5ii, 5iii, 5iv

All tests for this version have been passed therefore it should meet the stakeholder requirements.

Frequency Analysis

Frequency Analysis Version 1.0

First, I imported the Caesar Cipher encryption and decryption programs as well as the English checker as I was already aware that these would need to be used.

Brief walkthrough of the program:

1. Opens and reads the dictionary lines as a list.
2. sixMostFrequentLetters is defined as a list of constants.
3. tableOfCipherText is defined as a dictionary with every letter of the alphabet as a key and all values set to zero.
4. The ciphertext is defined within the program as a variable.
5. All ciphertext becomes lowercase.
6. mostfrequentletter() is called from within the variable definition of MFL
7. Loops through ciphertext and for every letter present it increments the value of the dictionary for that letter.
8. Then gets the most frequent variable through finding the maximum value for the values and returning the key as well.
9. Returns the key for the max value.
10. Letter is returned into MFL.
11. The attempting6x() function is called.
12. Loops 6 times, each time trying a different value for the most frequent letter from the list as the key.
13. It will use this key to attempt decryption.
14. Then splits the decryption into a list of words
15. Then puts this list of words through the English checker and the output is returned as a variable attemptEEng.
16. If the variable is true:
    a. Prints attemptEEng.
    b. Breaks the for loop.
17. If variable is false;
    a. Prints “not found”
    b. Continues through loop by incrementing the letter index.

Success Criteria Aim:

7ii

```
##Example text: Her last smile to me wasn't a sunset. It was an eclipse, t
he last eclipse, noon dying away to darkness where there would be no dawn.
import CaesarV5 as CC
import EnglishCheckerLevel1 as EC
a = open("english2.txt", "r")       #from english checker
with open("english2.txt", "r") as dictionary:
    a = dictionary.read().splitlines()

sixMostFrequentLetters=["e","t","a","o","i","n"]
```

```
tableOfCipherText = {"a":0,"b":0,"c":0,"d":0,"e":0,"f":0,
                     "g":0,"h":0,"i":0,"j":0,"k":0,"l":0,"m":0,
                     "n":0,"o":0,"p":0,"q":0,"r":0,"s":0,
                     "t":0,"u":0,"v":0,"w":0,"x":0,"y":0,"z":0}

#ciphertext is initial encrypted data, commented out code below is the
same as Example text
#ciphertext = "Nkx rgyz ysork zu sk cgyt'z g yatykz. Oz cgy gt kirovyk, zn
k rgyz kirovyk, tuut jeotm gcge zu jgxqtkyy cnkxk znkxk cuarj hk tu jgct."
ciphertext=ciphertext.lower()

def mostfrequentletter(text,dictionary):
    for letter in text:
            if letter.isalpha()==True:
                dictionary [letter]=dictionary[letter]+1
    mostfrequent=max(zip(dictionary.values(),dictionary.keys()))
    return mostfrequent[1]
#function called and result assigned to MFL variable
MFL=mostfrequentletter(ciphertext,tableOfCipherText)


def attempting6x(listSixMostFrequentLetters,modalLetter):
        letterindex=0
        while letterindex<6:
            attemptKey=ord(MFL)-ord(listSixMostFrequentLetters
[letterindex])

            attempt = CC.Caesar(ciphertext, attemptKey)
            decrypted = attempt.CaesarCipherDecryption()
            wordlist= EC.processTextLCaWS(decrypted).split()
            attemptEEng = EC.englishcheckerBF(wordlist,a)
            if attemptEEng==True:
                print(attemptEEng)
                break
            else:
                print("not found")
                letterindex+=1

attempting6x(sixMostFrequentLetters,MFL) #calling function
```

`mostfrequentletter(text, dictionary)` works out and returns only the most frequent letter of the text that has been given.

`attempting6x(listSixMostFrequentLetters, modalLetter)` loops through all the most likely possibilities until it finds the correct answer and then breaks the loop and prints the value of attemptEEng.

Problems with Version 1.0:

- The ciphertext is defined as a variable within the program.
- It isn't in a class.

Text 1: Pa tbza ohcl illu hyvbuk h xbhyaly av lslclu. H zhpsvy jhtl pu huk vyklylk h jopsl kvn huk jvmmll. P zspjlk h ibu, qlyrlk h myhur vba vm aol ivpspun dhaly, ulzalk pa, wvbylk h ohsm-kpwwly vm jopsl vcly aol myhur huk zwypurslk pa spilyhssf dpao jovwwlk vupvuz. P zjypiislk h joljr huk wba pa if opz wshal. P dvbsku'a ohcl yljvttluklk aol buwhshahisl tlzz av h zahycpun hupths. Aol zhpsvy dhz aol vusf jbzavtly, huk hmaly ol hal opz kvn ol slma. Aoha dhz aol lehja tvtlua zol lualylk. H zthss dvthu, ohyksf tvyl aohu mpcl mlla. Zol ohk aol mpnbyl vm h alluhnl npys. Oly zbpa dhz h isbl adllk, zthyasf jba, huk vcly oly aopu zovbsklyz zol dvyl h mby qhjrla, ivslyv slunao. Apuf nvsk jpyjbshy lhyypunz jsbun av oly zthss wplyjlk lhyz. Oly ohukz huk mlla dlyl zthss, huk dolu zol zlhalk olyzlsm ha aol jvbualy P uvapjlk zol dhzu'a dlhypun huf ypunz.
Text 2: It must have been around a quarter to eleven. A sailor came in and ordered a chile dog and coffee. I sliced a bun, jerked a frank out of the boiling water, nested it, poured a half-dipper of chile over the frank and sprinkled it liberally with chopped onions. I scribbled a check and put it by his plate. I wouldn't have recommended the unpalatable mess to a starving animal. The sailor was the only customer, and after he ate his dog he left. That was the exact moment she entered. A small woman, hardly more than five feet. She had the figure of a teenage girl. Her suit was a blue tweed, smartly cut, and over her thin shoulders she wore a fur jacket, bolero length. Tiny gold circular earrings clung to her small pierced ears. Her hands and feet were small, and when she seated herself at the counter I noticed she wasn't wearing any rings.[12]

[12] *Pick-Up – Charles Willeford*

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that it can crack a standard sentence.	Standard Data The test data is input through the program code as a variable.	First Chapter of Pick Up – Charles Willeford encrypted by the integer, 7 Refer to Text 1	Refer to Text 2	Brute Force: Text is likely to be English Over 90% of the words are English True	Fail

Version 1.0 Review

Success Criteria Met:

N/A

- The main reason for the success criteria not being met was because the program didn't print the decryption. Instead it printed True, which is useful to know it cracked it, but defeats the purpose as the stakeholder needs to be able to see what was cracked.
- Problems with Test Number 1 are remedied in Version 2.0.

Frequency Analysis Version 2.0 (Version in use)

In this version, the last program was turned into a class as well as the text that was the result of the cracking algorithm is returned in this version.

This algorithm works by compiling a dictionary of every letter in the alphabet and the most frequently occurring letter gets compared to 'e' first and then the next 5 most frequent letters after that and then using the English checker it will return the output that is in English or it will return 'not found'.

Success Criteria:

7ii, 7iii

```
##Cracking Caesar Cipher: Frequency Analysis

import CaesarV5 as CC
import EnglishCheckerFinal as EC

class frequencyAnalysis:
    def __init__(self,ciphertext):
        self.ciphertext = ciphertext.lower()
        self.tableOfCipherText = {"a":0,"b":0,"c":0,"d":0,"e":0,"f":0,
                    "g":0,"h":0,"i":0,"j":0,"k":0,"l":0,"m":0,
                    "n":0,"o":0,"p":0,"q":0,"r":0,"s":0,
                    "t":0,"u":0,"v":0,"w":0,"x":0,"y":0,"z":0}
        self.sixMostFrequentLetters=["e","t","a","o","i","n"]
        self.dictionary = open("english2.txt", "r").read().splitlines()

    def modalLetter(self):
        for letter in self.ciphertext:
                if letter.isalpha()==True:
                    self.tableOfCipherText[letter]=self.tableOfCipherText[
letter]+1
        mostfrequent=max(zip(self.tableOfCipherText.values(),self.tableOfC
ipherText.keys()))
        MFL = mostfrequent[1]
        return MFL

    def attemptingtocrack(self,modal):
        letterindex=0
        correctDecryption = "not found"
        while letterindex<6:
            attemptKey=ord(modal)-
ord(self.sixMostFrequentLetters[letterindex])
            ##print(attemptKey)
            attempt = CC.Caesar(self.ciphertext, attemptKey)
            attempting = attempt.CaesarCipherDecryption()
```

```
            ##print(attempting)  ##prints decrypted text regardless of whe
ther it is correct
            wordlist= EC.EnglishChecker(attempting)
            attemptEEng = wordlist.englishcheckerBF()
            ##print(attemptEEng)  ##True or False
            if attemptEEng==True:
                correctDecryption=attempting
                letterindex+=1      ##It will continue running through the
 list so the while loop is satisfied
            else:
                letterindex+=1
        return(correctDecryption)

    def runFA(self):
        MFL = self.modalLetter()
        cracked=self.attemptingtocrack(MFL)
        return cracked
##initial test data
#ciphertext = "Nkx rgyz ysork zu sk cgyt'z g yatykz. Oz cgy gt kirovyk, zn
k rgyz kirovyk, tuut jeotm gcge zu jgxqtkyy cnkxk znkxk cuarj hk tu jgct."
#ciphertext = "g hxuct cuuj ingox, haorz lxus ugq gtj gyn, oy huxotm. eua
ynuarj vgotz oz hrgiq gtj otjomu."
#test = frequencyAnalysis(ciphertext)
#print(test.runFA())
```

Code from GUI about implementation of frequency analysis within the Caesar Cipher section of the GUI

```
import frequencyAnalysisFinal as FA      ##This is Caesar Cipher Cracking

. . .

CCtext=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 9),text="E
nter text to be processed:").grid(row=9)
entry2 = Entry(caesar_frame)
entry2.grid(row = 9, column = 1)

crackButton = Button(caesar_frame, bg="white",font=("Helvetica", 9), text = "C
rack", command = lambda: self.CCCrack(caesar_frame,entry2))
crackButton.grid(row = 11, column = 2)
```

frequencyAnalysis
Text
runFA()

Test

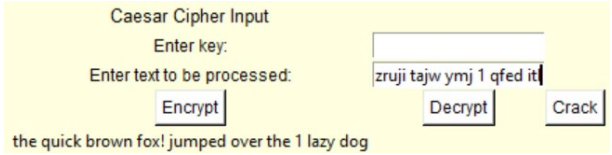
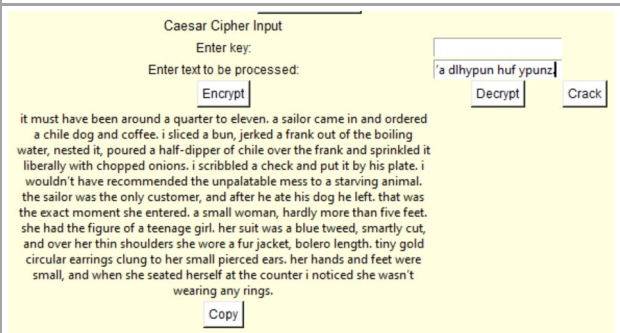
Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that it can decipher an encrypted piece of text without the key.	Standard data (Entered through the GUI)	Encrypted Text: ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl	The quick brown fox! Jumped over the 1 lazy dog		Pass
2	Checking that a longer piece of cipher text can be cracked.	Standard data (Entered through the GUI)	Encrypted Text: Refer to Text 1 in Version 1.0	Refer to Text 2 in Version 1.0		Pass

Table 7 Testing Frequency Analysis V_._

Version 3.0 Review

Success Criteria Met:

7ii, 7iii

All tests for this version have been passed therefore it should meet the stakeholder requirements. .

Baconian Cipher

Baconian Cipher Version 1.0

The primary aim of this first version was to make a working Baconian Cipher encryption algorithm.

Brief walkthrough of program:

1. A preset table is defined with alphabetical keys and the A or B values.
2. ABstring is a variable initialized as an empty list.
3. A flag variable is set to False.
4. Loops through a while loop with the condition that isn't satisfied until the flag is True.
5. Takes a string input.
6. Converts the string input into lowercase letters.
7. Loops through a for loop the length of the string input.
8. If the letter is j, it will replace it with the AB value of i.
9. If the letter is v, it will replace it with the AB value of u.
10. Otherwise it will append the AB value of the indexed letter.

Success Criteria Aim:

8ii

```
##Baconian Cipher Encrypt
##a is lowercase, b is capital
##using only i and u (if statement will be needed for j and v)

import random
import string

ABvalues= {"a":"aaaaa","b":"aaaab","c":"aaaba","d":"aaabb","e":"aabaa","f"
:"aabab","g":"aabba","h":"aabbb","i":"abaaa","k":"abaab","l":"ababa","m":"
ababb","n":"abbaa","o":"abbab","p":"abbba","q":"abbbb","r":"baaaa","s":"ba
aab","t":"baaba","u":"baabb","w":"babaa","x":"babab","y":"babba","z":"babb
b"}

##ABstring="" <-- Wasnt working
ABstring = []

flag = False
while flag != True:
    letter = input(str("What letter: ")) ##needs while loop for validation
    letter = letter.lower()
    for i in range(len(letter)):
        if letter[i].isalpha():
            if str(letter[i]) == "j":
                changeto="i"
                ABstring.append(ABvalues[letter["i"]])
            if str(letter[i]) == "v":
```

```
                changeto="u"
            else:
                ABstring += ABvalues[letter[i]]
    done = input("Done?: ")
    if done == "y":
        flag=True
    else:
        flag=False

randomtext=""

for aOrB in ABstring:
    if aOrB =="a":
        loweralpha=string.ascii_lowercase
        randomletter = random.choice(loweralpha)

        randomtext += randomletter
    else:
        upperalpha=string.ascii_uppercase
        randomletter = random.choice(upperalpha)
        randomtext += randomletter

print(randomtext)
```

Problems with Version 1.0:

- All user interaction is within the terminal.
- It is not in class format.
- It is coded fully using a procedural paradigm as there is no use of functions.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that the program can successfully encrypt using the Baconian cipher.	Standard Data (Entered through the terminal)	Text: finding	Random letters with cases as followed: aababababaaaabbaaa-aabbabaaaabbaaaabba	What letter: finding Done?: y esBbRnKxdqlIPtvnhdKYnUjyjqKKufydAFg	Pass

Version 1.0 Review

Success Criteria Met:

8ii

All tests for this version have been passed therefore it should meet the stakeholder requirements. However it does not fit with the rest of the program so will still need further versions to adapt it.

Baconian Cipher Version 1.1

Not much changed since the last version but this version has the beginnings of turning parts of the code into functions.

Success Criteria Aim:

8ii

```
##Baconian Cipher Encrypt
#a is lowercase, b is capital
##Problem: Will not accept the i and j switch along with the other one
##using only i and u (if statement will be needed for j and v)

import random
import string

ABvalues= {"a":"aaaaa","b":"aaaab","c":"aaaba","d":"aaabb","e":"aabaa","f"
:"aabab","g":"aabba","h":"aabbb","i":"abaaa","k":"abaab","l":"ababa","m":"
ababb","n":"abbaa","o":"abbab","p":"abbba","q":"abbbb","r":"baaaa","s":"ba
aab","t":"baaba","u":"baabb","w":"babaa","x":"babab","y":"babba","z":"babb
b"}

##print(ABlist["a"])
##ABstring="" <-- Wasnt working

ABstring = []

flag = False
while flag != True:
    letter = input(str("What letter: ")) ##needs while loop for validation
    letter = letter.lower()
    for i in range(len(letter)):
        if letter[i].isalpha():
            print(letter[i])
            print(type(letter[i]))
            if str(letter[i]) == "j":
                print("working?")  ##Checking if the if loop is used
                changeto="i"
                ##ABstring += ABvalues[letter(changeto)]
                ABstring.append(ABvalues[letter["i"]])
            if str(letter[i]) == "v":
                changeto="u"
                ##ABstring += ABvalues[letter(changeto)]
            else:
                ABstring += ABvalues[letter[i]]
    done = input("Done?: ")
    if done == "y":
        flag=True
    else:
```

```
        flag=False

def randtext(ABnfive):
    randomtext=""
    for aOrB in ABnfive:
        if aOrB =="a":
            loweralpha=string.ascii_lowercase
            randomletter = random.choice(loweralpha)
            randomtext += randomletter
        else:
            upperalpha=string.ascii_uppercase
            randomletter = random.choice(upperalpha)
            randomtext += randomletter
    return randomtext

randtext(ABString)

print(randomtext)
```

The `randtext(ABnfive)` function produces random text with capital letters and lowercase letters corresponding to what has been passed through the function.

Problems with Version 1.1:

- It is not in class format.
- All the user interaction is in the terminal.
-

Baconian Cipher Version 2.0

The aim of this version was to make a decryption program. This has been created as a class called Baconian because I was able to think ahead this time. The encryption program can quite easily be added now that the decryption program is in the form of a class.

Walkthrough of the program:

1. Analyses by looping through all the entered input to make a list of a's and b's depending on whether the letter in question is lowercase, in which a is appended to the list or the letter being uppercase in which b is appended to the list.
2. It then creates a list with every 5 letters concatenated to make a list of 5 letter strings.
3. It then goes through this list, using the ABValues dictionary in order to identify what letter is encoded and decode it and add it to the decrypted answer.
4. It then returns the decrypted answer.

Success Criteria Aim:

9ii

```
##Baconian Cipher Decryption
##using only i and u (if statement will be needed for j and v)
```

```
import random
import string
class Baconian:
    def __init__(self, text):
        self.text = list(text)
        self.ABvalues = {"a":"aaaaa","b":"aaaab","c":"aaaba","d":"aaabb","e":"aabaa","f":"aabab",
            "g":"aabba","h":"aabbb","i":"abaaa","k":"abaab","l":"ababa","m":"ababb",
            "n":"abbaa","o":"abbab","p":"abbba","q":"abbbb","r":"baaaa","s":"baaab",
            "t":"baaba","u":"baabb","w":"babaa","x":"babab","y":"babba","z":"babbb"}
        self.ABkeys = {}
        for key, value in (self.ABvalues).items():
            self.ABkeys[value] = key


##print(ABlist["a"])

    def analyseFiveLetters(self):
        upperAndLower = []
        for char in (self.text):
            if char.isalpha():
                if char.islower():
                    upperAndLower.append("a")
                elif char.isupper():
                    upperAndLower.append("b")
        return upperAndLower

    def listOfAB(self,upperAndLower):
        listOfFive = []
        for n in range (int((len(upperAndLower))/5)):
            fiveLetters = []
            fiveLetters.append(upperAndLower[5*n])
            fiveLetters.append(upperAndLower[5*n+1])
            fiveLetters.append(upperAndLower[5*n+2])
            fiveLetters.append(upperAndLower[5*n+3])
            fiveLetters.append(upperAndLower[5*n+4])
            listOfFive.append("".join(fiveLetters))
        return listOfFive

    def comparison(self,listOfFive):
        listOfDecryption=[]
        for ABcode in listOfFive:       ##goes through ab list from input
            for ABalpha in (self.ABkeys).keys():    ##ABalpha is a key in thhe ABKEys
                if ABcode==ABalpha:
                    listOfDecryption.append(self.ABkeys[ABalpha])
```

```
        decrypted="".join(listOfDecryption)
        return decrypted

    def BaconianCipherDecryption(self):
        baconianObject = Baconian(self.text)  ##returns with B
        a=baconianObject.analyseFiveLetters()
        b=baconianObject.listOfAB(a)
        return baconianObject.comparison(b)

##def TestDecrypt():
##    bacTest = Baconian("aaaBB")
##    print(bacTest.BaconianCipherDecryption())
##
##TestDecrypt()
```

The `analyseFiveLetters(self)` function creates a list of a's and b's corresponding to the case of the input while its being iterated through.

The `listOfAB(self,upperAndLower)` function creates strings of 5 characters and appends them to a list and returns the list.

The `comparison(self,listOfFive)` function iterates through and identifies what letter the 5 letter string's corresponding alphabetical value is and appends this to a decrypt list and then turns this to a string.

The `BaconianCipherDecryption(self)` function runs all the subprograms necessary.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that the program can successfully decrypt using the Baconian cipher.	Standard Data (Entered through the object)	Random letters with cases as followed: aababababaaaabbaaa-aabbabaaaabbaaaabba	finding	finding	Pass

Version 2.0 Review

Success Criteria Met:

9ii

All tests for this version have been passed therefore it should meet the stakeholder requirements.

Baconian Cipher Version 3.0 (Version in use)

For this cipher there are two separate algorithms for encryption and decryption because unlike the shift ciphers, this is modelled as taking data in different formats for encrypting and decrypting. In this version I combine those ciphers and put them into one class so they can be called through the class 'Baconian'.

Success Criteria Aim:

8ii, 8iii, 9ii, 9iii

```
##using only i and u
import random
import string  ##forgot to import module 07022020

class Baconian:
    def __init__(self, text):
        self.text = list(text)
        self.ABvalues = {"a":"aaaaa","b":"aaaab","c":"aaaba","d":"aaabb","
e":"aabaa","f":"aabab",
            "g":"aabba","h":"aabbb","i":"abaaa","k":"abaab","l":"ababa","m
":"ababb",
            "n":"abbaa","o":"abbab","p":"abbba","q":"abbbb","r":"baaaa","s
":"baaab",
            "t":"baaba","u":"baabb","w":"babaa","x":"babab","y":"babba","z
":"babbb"}
        self.ABkeys = {}
        for key, value in (self.ABvalues).items():
            self.ABkeys[value] = key

##Encryption Algorithms

    def makingAB(self):
        ABstring = []
        letter = self.text
        for character in range(len(letter)):
            if letter[character].isalpha():
                ##print(letter[i])
                ##print(type(letter[i]))
                if str(letter[character]) == "v":
                    changeto="u"
                    ABstring += self.ABvalues[changeto]
                elif str(letter[character]) == "j":
                    changeto="i"
                    ABstring += self.ABvalues[changeto]
                else:
                    ABstring += self.ABvalues[letter[character]]
            else:
```

```
                ABstring += character
        return ABstring
        #print(ABvalues[letter])

    #a is lowercase, b is capital

    def randtext(self,ABnfive):
        randomtext=str("")
        for aOrB in ABnfive:
            if aOrB =="a":
                loweralpha=string.ascii_lowercase
                randomletter = random.choice(loweralpha)
                randomtext += randomletter
            else:
                upperalpha=string.ascii_uppercase
                randomletter = random.choice(upperalpha)
                randomtext += randomletter
        return randomtext

    def BaconianCipherEncryption(self):
        encyptedText = self.makingAB()
        return self.randtext(encyptedText)

##Decryption Algorithms

    def analyseFiveLetters(self):
        upperAndLower = []
        for char in (self.text):
            if char.isalpha():
                if char.islower():
                    upperAndLower.append("a")
                elif char.isupper():
                    upperAndLower.append("b")
        return upperAndLower

    def listOfAB(self,upperAndLower):
        listOfFive = []
        for n in range (int((len(upperAndLower))/5)):
            fiveLetters = []
            fiveLetters.append(upperAndLower[5*n])
            fiveLetters.append(upperAndLower[5*n+1])
            fiveLetters.append(upperAndLower[5*n+2])
            fiveLetters.append(upperAndLower[5*n+3])
            fiveLetters.append(upperAndLower[5*n+4])
            listOfFive.append("".join(fiveLetters))
        return listOfFive

    def comparison(self,listOfFive):
```

```
        listOfDecryption=[]
        for ABcode in listOfFive:       ##goes through ab list from input
            for ABalpha in (self.ABkeys).keys():    ##ABalpha is a key in
thhe ABKEys
                if ABcode==ABalpha:
                    listOfDecryption.append(self.ABkeys[ABalpha])
        decrypted="".join(listOfDecryption)
        return decrypted

    def BaconianCipherDecryption(self):
        baconianObject = Baconian(self.text)  ##returns with B
        a=baconianObject.analyseFiveLetters()
        b=baconianObject.listOfAB(a)
        return baconianObject.comparison(b)



##def TestDecrypt():
##    bacTest = Baconian("aaaBB")
##    print(bacTest.BaconianCipherDecryption())
##
##TestDecrypt()
```

Code from GUI about Baconian Cipher within the GUI

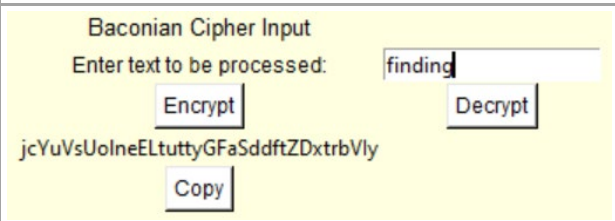
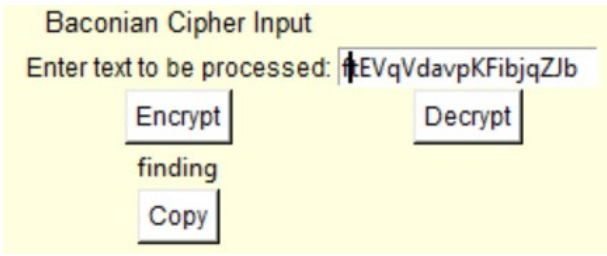
This part of the program is mainly initialising the frames for the Baconian cipher

```
import BaconianVer0 as BC
      . . .
self.baconian_frame = Frame(root,bg="Light Yellow")
self.baconian_frame.pack()
self.baconian_result = Label(self.baconian_frame, bg= "Light Yellow", wrapleng
th=400)
self.baconian_result.grid(row=14)
      . . .


secondbutton = Button(self.main_frame, bg="white",font=("Helvetica", 9),text="
Baconian Cipher", command=lambda: self.BaconianSelect()).grid(row= 3, column =
 0)
      . . .
    def BaconianSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack()
        self.vigenere_frame.pack_forget()
        self.baconian_result.config(text="")  ##reseting the result value
        baconian_frame=self.baconian_frame
```

```
        BCnote=Label(self.baconian_frame, bg="Light Yellow",font=("Helvetica",
 10), text="Baconian Cipher Input").grid(row=7, column=0)
        entry=Entry(self.baconian_frame)
        BCtext=Label(self.baconian_frame, bg="Light Yellow",font=("Helvetica",
 9),text="Enter text to be processed:").grid(row=8)

        encryptButton = Button(self.baconian_frame, bg="white",font=("Helvetic
a", 9), text = "Encrypt",command=lambda: self.BCEncryption(entry))
        decryptButton = Button(self.baconian_frame, bg="white",font=("Helvetic
a", 9), text = "Decrypt",command=lambda: self.BCDecryption(entry))
        encryptButton.grid(row=11)
        decryptButton.grid (row = 11, column = 1)
        entry.grid(row=8,column=1)


    def BCEncryption(self,entry):
        text = entry.get()
        baconianE = BC.Baconian(text)
        encrypting=baconianE.BaconianCipherEncryption()
        self.baconian_result.config(text=encrypting)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(encrypting))
        copytoclipboard.grid(row=15)

    def BCDecryption(self, entry):
        text = entry.get()
        baconianD = BC.Baconian(text)
        decrypting=baconianD.BaconianCipherDecryption()
        self.baconian_result.config(text=decrypting)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(decrypting))
        copytoclipboard.grid(row=15)
```

Baconian
Text
BaconianCipherEncryption()
BaconianCipherDecryption()

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that the program can successfully encrypt using the Baconian cipher.	Standard Data (Entered through the GUI)	Text: finding	Random letters with cases as followed: aababababaaaabbaaa-aabbabaaaabbaaaabba		Pass
2	Checking that the program can successfully decrypt using the Baconian cipher.	Standard data	Text: neFgVyWytgoNHklwf-tEVqVdavpKFibjqZJb	Decrypted Text: finding		Pass

Version 3.0 Review

Success Criteria Met:

8ii, 8iii, 9ii, 9iii

All tests for this version have been passed therefore it should meet the stakeholder requirements.

Vigenère Cipher

Vigenère Cipher Version 1.0

First, I worked on encryption as a standalone program as I was aware that to create a decryption program it would only require minor changes to the shift. This is the first version of the Vigenère cipher encryption that produces an output that is meant to be the encrypted text.

Brief walkthrough of the program:

1. Receives two inputs from the terminal for the key and the text to be encrypted.
2. Validates that the key is in an acceptable format.
3. Converts text into a text list of all characters.
4. Converts key into a number list with corresponding letters of the alphabet.
5. Goes through list of characters and applies the shift using a loop for every alphabetical character.
6. Produces the output

The primary aim of this first version was to complete a Vigenère cipher shift that works regardless of any formatting errors.

Success Criteria Aim:

10ii, 10iii

```
#This will be the Vigenere cipher's data
def VCData():
    ##key being validated and processed
    validateKey=False
    while validateKey==False:
        key= input("Please enter key: ")       #key will need validation o
f being a single word
        #global keylist
        keytemplist = list(key)
        for letter in keytemplist:
            if letter.isalpha()==False:
                validateKey=False
                print("Please enter only one word without any numbers, spa
ces or special characters.")
                break
            else:
                validateKey=True
        print("Is key in an acceptable format?",validateKey)    #will only
 show if key is in correct format now
    keylist=processtext(keytemplist)
    ##text being processed, so that it is all lowercase
    plaintext= input("Please enter the text to be encrypted: ")
    global textlist
    templist=list(plaintext)
    ##making all capital letters lowercase
```

```
    textlist=processtext(templist)
    global shiftlist
    shiftlist=[]
    for alpha in keylist:
        keynum = keyshift(alpha)
        shiftlist.append(keynum)
    ##print(shiftlist)

def processtext(templist):
    textlist=[]
    lowerA=ord("a")
    capitalA=ord("A")
    capToLowConstant=lowerA-capitalA
    for i in templist:
        if i.isalpha()==True:
            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
                j=chr((ord(i)+capToLowConstant))
                textlist.append(j)
            else:
                 textlist.append(i)
        else:
            textlist.append(i)
    return textlist     ##textlist is a list of of the chatacters in the t
ext


def keyshift(keylisted):       #This is used to make the numerical key for
 Vigenere
    shift=ord(keylisted)-96
    #This will make it so a = 1, b = 2, c = 3, ..., z = 26
    return shift

def VCEncrypt(letterlist,numberlist):
    encryptlist=[]
    for a in letterlist:
        if a.isalpha()==True:   ##validates that letter is in the alphabet
            for number in numberlist:
                shiftedletter=convert(a,number)
                encryptlist.append(shiftedletter)
        else:
            encryptlist.append(a)
    joined="".join(map(str,encryptlist))
    print ("The following is the encrypted text:\n"+joined)    ##making th
e output part of a string

def convert(letter, shift):   ##same as used in the Caesar Cipher
     l=chr(((ord(letter)-96)+shift)%26+96)
     return l
```

```
def VigenereCipherEncryption():
    print("Vigenere Cipher > Encryption")
    VCData()
    VCEncrypt(textlist,shiftlist)

VigenereCipherEncryption()
```

`VCData()` validates the key by using a while loop and while the key is not in the right format, it will repeatedly as the user for the right input for the key until it is given a key in the correct format. It then makes the text given a list and then creates a keylist, which is all the letters in a corresponding numerical format.

`Processtext(templist)` is the same as the Caesar Cipher and makes all characters in the text given lowercase.

`Keyshift(keylisted)` takes the given letter and converts it into a numerical value.

`VCEncrypt(letterlist)` goes through the list and if the letter is alphabetical then it will apply the shift, which also is in a loop.

`VigenereEncryption()` makes it so that `VCData()` runs first and then `VCEncrypt()` runs with the outputs of `VCData()`.

Problems with Version 1.0:

- Use of global variables within a function.
- All user interaction within the terminal.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that a word can be successfully encrypted using the Vigenère cipher.	Standard Data (Entered through the terminal)	Key: abc Text: the	tig	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the The following is the encrypted text: uvwijkfgh	Fail
2	Checking that one lowercase sentence can be encrypted. (Checks spaces are unchanged)	Standard Data -	Key = abc Text = the quick brown fox jumped over the lazy dog	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the quick br own fox jumped over the lazy dog The following is the encrypted text: uvwijkfgh rstvwxjkldeflmn cdestupqrxy`opq ghipqry`a klmvwxnopqrsfghefg pqrwxyfghstu uvwijkfgh mnobcdab c`ab efgpqrhij	Fail
3	Checking that one mixed case sentence can be encrypted.	Standard Data -	Key = abc Text = The QUick brown FOX jumped over the lazy doG	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: The QUick brown FOX jumped over the lazy doG The following is the encrypted text: uvwijkfgh rstvwxjkldeflmn cdestupqrxy`opq ghipqry `a klmvwxnopqrsfghefg pqrwxyfghstu uvwijkfgh mnob cdabc`ab efgpqrhij	Fail

4	Checking that one uppercase sentence can be encrypted.	Standard Data -	Key = abc Text = THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG The following is the encrypted text: uvwijkfgh rstvwxjkldeflmn cdestupqrxy`opq ghipqry `a klmvwxnopqrsfghefg pqrwxyfghstu uvwijkfgh mnob cdabc`ab efgpqrhij	Fail
5	Checking that numbers and special characters are not affected.	Standard Data -	Key: abc Text: the & the 1	tig & tig 1	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the & the 1 The following is the encrypted text: uvwijkfgh & uvwijkfgh 1	Fail (or pass with wrong output for 'the')

Table 8 Testing Vigenère Cipher V1.0

Version 1.0 Review

Success Criteria Met:

N/A

On first observation, I can see that the output text is much longer than expected so it isn't encrypting the text correctly.

It returns the character ` in places so that needs to be rectified like in the Caesar Cipher.

Seems to be shifting by the wrong amount initially because in the Vigenère, t=t when shifted by a.

Vigenère Cipher Version 1.1

The only changes made to the last version are in the VCEncrypt() and convert() functions.

One of the aims of this version is to rectify the incorrect shift. As well as stop the output from producing a string which was a lot longer than expected.

Success Criteria Aim:

10ii, 10iii

```
##This is for Vigenere
def VCEncrypt(letterlist,numberlist):
    encryptlist=[]
    for character in letterlist:
        if character.isalpha()==True:
            ##validates that letter is in the alphabet
            shiftedletter=convert(character,(numberlist[letterlist.index(c
haracter)%len(numberlist)]-1))    ##Added -1 to rectify the shift
            encryptlist.append(shiftedletter)
        else:
            encryptlist.append(a)
    joined="".join(map(str,encryptlist))


def convert(letter, shift):   ##same as used in the Caesar Cipher
    l=chr(((ord(letter)-96)+shift)%26+96)
    if l =="`":
        l="z"
    return l
```

The VCEncrypt() function has had a for loop removed because this was producing a longer output than expected.

The convert() function like in the Caesar Cipher now has an exception for the character to replace it with the right letter.

Problems with Version 1.1:

- User interaction within the terminal.
- Use of global variables within functions.
- It's not using objected oriented programming, such as classes.

Vigenère Cipher Version 2.0

I approached this cipher slightly differently from the Caesar and Baconian cipher because I changed the encryption program so it will encrypt and decrypt by changing the direction of the shifts so that it the decryption part is just added to the encryption part and there is different ways of calling the encryption and decryption algorithms.

Success Criteria Aim:

10ii, 10iii, 11ii, 11iii

```
def VCData():                 #This will be the vigenere cipher's data
    ##key being processed
    validateKey=False
    while validateKey==False:
        key= input("Please enter key: ")            #key will need validat
ion of being a single word
        keytemplist = list(key)
        for letter in keytemplist:
            if letter.isalpha()==False:
                validateKey=False
                print("Please enter only one word without any numbers, spa
ces or special characters.")
                break
            else:
                validateKey=True
        print("Is key in an acceptable format?",validateKey)        #will
only show if key is in correct format now
    keylist=processtext(keytemplist)
    ##text being processed
    plaintext= input("Please enter the text to be encrypted: ")
    templist=list(plaintext)
    ##making all capital letters lowercase
    textlist=processtext(templist)
    #creating a list with the shift
    shiftlist=[]
    for alpha in keylist:
        keynum = keyshift(alpha)
        shiftlist.append(keynum)
    return textlist, shiftlist

##now successfully returns a list with the shifts in
##needs to be taken out and replaced with same code as the CCFull
def processtext(templist):
    textlist=[]
    lowerA=ord("a")
    capitalA=ord("A")
    capToLowConstant=lowerA-capitalA
```

```
    for i in templist:
        if i.isalpha()==True:
            if ord(i)<=ord("Z") and ord(i)>=ord("A"):
                j=chr((ord(i)+capToLowConstant))
                textlist.append(j)
            else:
                 textlist.append(i)
        else:
            textlist.append(i)
    return textlist


def keyshift(keylisted):
    shift=ord(keylisted)-96
#this will make it so a = 1, b = 2, c = 3, ..., z = 26
    return shift


def VCChange(letterlist,numberlist,rightorleft):   ##This is for Vigenere
    encryptlist=[]
    for character in letterlist:
        if character.isalpha()==True:   ##validates that letter is in the
alphabet
            shiftedletter=convert(character,(numberlist[letterlist.index(c
haracter)%len(numberlist)]-
1), rightorleft)  ##second part is just the shift
            encryptlist.append(shiftedletter)

        else:
            encryptlist.append(character)
    joined="".join(map(str,encryptlist))
    print ("The following is the encrypted text:\n"+joined)

def convert(letter, shift, rightorleft):
##same as used in the Caesar Cipher
    l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
    if l =="`":
        l="z"
    return l


def VigenereCipherEncryption():
    mode = +1
    print("Vigenere Cipher > Encryption")
    textlist, shiftlist = VCData()
    VCChange(textlist,shiftlist,mode)

def VigenereCipherDecryption():
    mode = -1
```

```
    print("Vigenere Cipher > Decryption")
    textlist, shiftlist = VCData()
    VCChange(textlist,shiftlist,mode)

VigenereCipherEncryption()
VigenereCipherDecryption()
```

The `VigenereCipherEncryption()` function makes the shift positive.

The `VigenereCipherDecryption()` function makes the shift negative.

Problems with Version 2.0:

- It's not using objected oriented programming, such as classes.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that a word can be successfully encrypted using the Vigenère cipher.	Standard Data (Entered through the terminal)	Key: abc Text: the	tig	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the The following is the encrypted text: tig	Pass
2	Checking that one lowercase sentence can be encrypted. (Checks spaces are unchanged)	Standard Data -	Key = abc Text = the quick brown fox jumped over the lazy dog	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the quick br own fox jumped over the lazy dog The following is the encrypted text: tig rwidm ctoxp gox lwnrge owgt tig lbby eoh	Fail
3	Checking that one mixed case sentence can be encrypted.	Standard Data -	Key = abc Text = The QUick brown FOX jumped over the lazy doG	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: The QUick br own FOX jumped over the lazy doG The following is the encrypted text: tig rwidm ctoxp gox lwnrge owgt tig lbby eoh	Fail
4	Checking that one uppercase sentence can be encrypted.	Standard Data -	Key = abc Text = THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG	tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: THE QUICK BR OWN FOX JUMPED OVER THE LAZY DOG The following is the encrypted text: tig rwidm ctoxp gox lwnrge owgt tig lbby eoh	Fail

5	Checking that numbers and special characters are not affected.	Standard Data -	Key: abc Text: the & the 1	tig & tig 1	Vigenere Cipher > Encryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: the & the 1 The following is the encrypted text: tig & tig 1	Pass
6	Checking that a word can be successfully decrypted using the Vigenère cipher.	Standard Data (Entered through the terminal)	Key: abc Text: tig	the	Vigenere Cipher > Decryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: tig The following is the encrypted text: the	Pass
8	Checking that one lowercase sentence can be decrypted. (Checks spaces are unchanged)	Standard Data -	Key = abc Text = tig qvkcl drpyn gqx kwmqgd pxes vhf naaa dpi	the quick brown fox jumped over the lazy dog	Vigenere Cipher > Decryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: tig qvkcl dr pyn gqx kwmqgd pxes vhf naaa dpi The following is the encrypted text: the ptkbj cppxl epx kwlpec pxcs the lzzz cph	Fail
9	Checking that one mixed case sentence can be decrypted.	Standard Data -	Key = abc Text = Tig QVkcl drpyn GQX kwmqgd pxes vhf naaa dpI	the quick brown fox jumped over the lazy dog	Vigenere Cipher > Decryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: Tig QVkcl dr pyn GQX kwmqgd pxes vhf naaa dpI The following is the encrypted text: the ptkbj cppxl epx kwlpec pxcs the lzzz cph	Fail
10	Checking that one uppercase sentence can be decrypted.	Standard Data -	Key = abc Text = TIG QVKCL DRPYN GQX KWMQGD PXES VHF NAAA DPI	the quick brown fox jumped over the lazy dog	Vigenere Cipher > Decryption Please enter key: abc Is key in an acceptable format? True Please enter the text to be encrypted: TIG QVKCL DR PYN GQX KWMQGD PXES VHF NAAA DPI The following is the encrypted text: the ptkbj cppxl epx kwlpec pxcs the lzzz cph	Fail

Table 9 Testing Vigenère Cipher V2.0

Version 2.0 Review

Success Criteria Met:

N/A

- It can encrypt and decrypt a single word, but not a sentence meaning that something is probably wrong with the fact the data has spaces in it as it must be offsetting the rest of the encryption.

Vigenère Cipher Version 3.0 (Version in use)

The previous error where the presence of spaces in the data to be encrypted or decrypted has been rectified as well as the Vigenère encryption and decryption algorithms are now in a class called Vigenere.

Success Criteria Aim:

10ii, 10iii, 10iv

```
class Vigenere:
    def __init__(self,key,text):
        self.fixedKey = [ord(o.lower())-96 for o in key]
        self.key = []       ##this will be key used for decryption or encr
yption
        self.text = list([m.lower() for m in text])
        keyLengthwr=0   ##with repeats
        for i in self.text:
            if i.isalpha():
                keyLengthwr+=1
            else:
                pass
        for j in range(0,keyLengthwr): ##key is going wrong
            self.key.append(ord(key[j % len(key)])-96)

    def VCChange(self,rightorleft):                ##This is for Vigenere
        letterlist = self.text
        numberlist = self.key
        encryptlist=[]
        n=0
        for i in range(0,len(letterlist)):  ##changes made here: goes thro
ugh letterlist and n only increments if the letter is alphabetical
            character = letterlist[i]
            if character.isalpha():   ##validates that letter is in the al
phabet
                keynum=numberlist[n]
                ##print(keynum)
                shiftedletter=self.convert(character,(keynum-
1), rightorleft)
                encryptlist.append(shiftedletter)
                n+=1
            else:
                encryptlist.append(character)
        joined="".join(map(str,encryptlist))
        ##print ("The following is the encrypted text:\n"+joined)
        return joined

    def convert(self,letter, shift, rightorleft):   ##same as used in the
Caesar Cipher
        l=chr(((ord(letter)-(ord('a')-1))+(rightorleft)*shift)%26+96)
```

```
        if l =="`":
            l="z"
        return l


    def VigenereCipherEncryption(self):
        mode = +1
        ##print("Vigenere Cipher > Encryption")
        return self.VCChange(mode)


    def VigenereCipherDecryption(self):
        mode = -1
        ##print("Vigenere Cipher > Decryption")
        return self.VCChange(mode)

##Initial test data
#test=Vigenere("abcd",First paragraph of Book Shantaram encrypted)
#print(test.VigenereCipherEncryption())
#print(test.VigenereCipherDecryption())
```

Code from GUI about Vigenère Caesar Cipher within the GUI

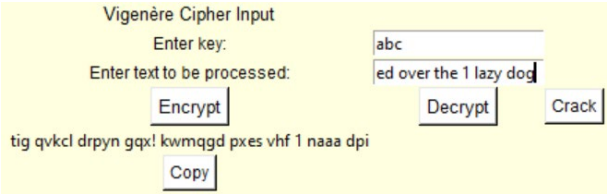
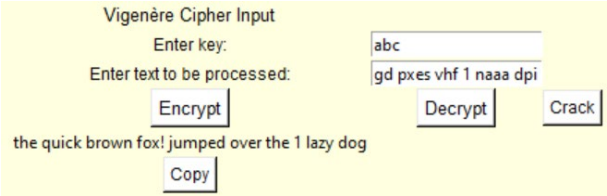
```
import VigenereV2 as VC

            . . .
self.vigenere_frame = Frame(root,bg="Light Yellow")
self.vigenere_frame.pack()
self.vigenere_result = Label(self.vigenere_frame, bg= "Light Yellow", wrapleng
th=400)
self.vigenere_result.grid(row=14)

            . . .
def VCEncryption(self,entry1,entry2):    ##Validation for key: string, no numb
ers, special characters or spaces
    key = entry1.get()
    if key.isalpha() == False:
        messagebox.showerror("Invalid Key","Please enter the key as a string o
f only letters.")
            entry1.delete(0,"end")
    text = entry2.get()
    vigenereE = VC.Vigenere(key,text)
    encrypting=vigenereE.VigenereCipherEncryption()
    self.vigenere_result.config(text=encrypting)
    copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica", 9
),text="Copy",command=lambda: cbt.copy(encrypting))
    copytoclipboard.grid(row=15)
```

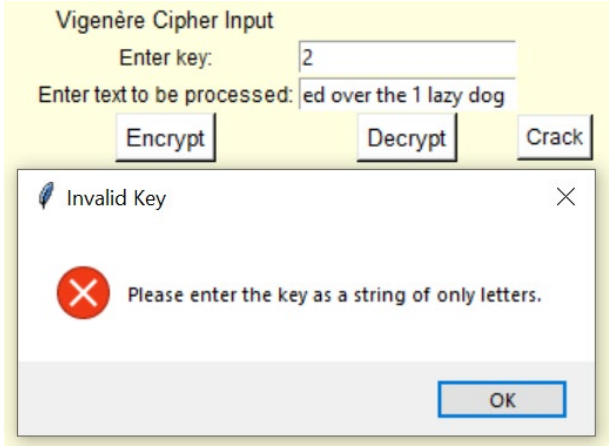
```
def VCDecryption(self,entry1,entry2):    ##Validation for key: string, no numb
ers, special characters or spaces
    key = entry1.get()
    if key.isalpha() == False:
        messagebox.showerror("Invalid Key","Please enter the key as a string o
f only letters.")
        entry1.delete(0,"end")
    text = entry2.get()
    vigenereD = VC.Vigenere(key,text)
    decrypting=vigenereD.VigenereCipherDecryption()
    self.vigenere_result.config(text=decrypting)
    copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica", 9
),text="Copy",command=lambda: cbt.copy(decrypting))
    copytoclipboard.grid(row=15)


def VCCrack(self,entry2):
    text = entry2.get()
    vigenereC = KA.VigenereCrack(text)
    cracking = vigenereC.VCCrack()
    self.vigenere_result.config(text = cracking)
    copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica", 9
),text="Copy",command=lambda: cbt.copy(cracking))
    copytoclipboard.grid(row=15)

            . . .
    def VigenereSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack()
        self.vigenere_result.config(text="")
        vigenere_frame=self.vigenere_frame

        VCnote=Label(vigenere_frame,bg="Light Yellow",font=("Helvetica", 10),
text="Vigenère Cipher Input").grid(row=7, column=0)
        VCkey=Label(vigenere_frame, bg="Light Yellow",font=("Helvetica", 9),te
xt="Enter key:").grid(row=8)
        entry1=Entry(vigenere_frame)

        VCtext=Label(vigenere_frame,bg="Light Yellow",font=("Helvetica", 9),te
xt="Enter text to be processed:").grid(row=9)
        entry2 = Entry(vigenere_frame)
        entry1.grid(row=8,column=1)
        entry2.grid(row = 9, column = 1)

        encryptButton = Button(vigenere_frame, bg="white",font=("Helvetica", 1
0), text = "Encrypt",command=lambda: self.VCEncryption(entry1,entry2))
        decryptButton = Button(vigenere_frame, bg="white",font=("Helvetica", 1
0), text = "Decrypt",command=lambda: self.VCDecryption(entry1,entry2))
```

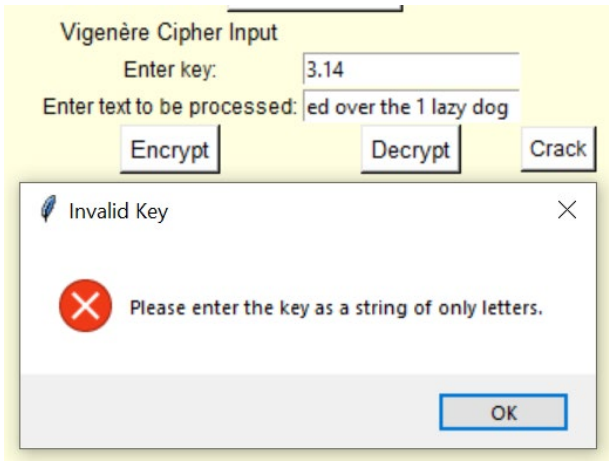
```
        crackButton = Button(vigenere_frame, bg="white",font=("Helvetica", 9),
 text = "Crack", command = lambda: self.VCCrack(entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid (row = 11, column = 2)

            . . .
thirdbutton = Button(self.main_frame, bg="white",font=("Helvetica", 9), text =
 "Vigenère Cipher", command=lambda: self.VigenereSelect()).grid(row= 4, column
 = 0)
```

Vigenere
Key
Text
VigenereCipherEncryption()
VigenereCipherDecryption()

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that a text can be successfully encrypted using the Vigenère cipher.	Standard Data (Entered through the GUI)	Key = abc Text = The Quick Brown Fox! jumped over the 1 lazy dog	tig qvkcl drpyn gqx! kwmqgd pxes vhf 1 naaa dpi		Pass
	Checking that a text can be successfully decrypted using the Vigenère cipher.		Key = abc Text = Tig Qvkcl Drpyn Gqx! kwmqgd pxes vhf 1 naaa dpi	the quick brown fox! jumped over the 1 lazy dog		Pass
	Checking that a non-string with only letters key creates an error message box for encryption/decryption.	Erroneous Data -	Vigenère Cipher button clicked Text = The Quick Brown Fox! jumped over the 1 lazy dog Key(s) = String longer than 1 word: “hello world”, Integer: 2, Float: 3.14	A message box appears to stop the input being accepted.	String longer than 1 word:	Pass

			Encrypt/Decrypt button clicked		 Integer:  Float:	

Version 3.0 Review

Success Criteria Met:

10ii, 10iii, 10iv, 11ii, 11iii, 11iv

All tests for this version have been passed therefore it should meet the stakeholder requirements.

Kasiski Analysis

Kasiski Analysis Version 1.0

My approach to this was to find the key length first. Then to brute force all possible combinations with this key length. Although this would mean a longer key would increase the time to complete this algorithm so that is a preconceived concern.

Brief walkthrough of the program:

1. Converts the cipher text to a format where there are only alphabetical characters with no spaces.
2. A subprogram for getting all the index positions of a certain substring is initialised, but not used yet.
3. It creates several groups of three, incrementing like (0,1,2), (1,2,3),… and every repeated occurrence is recorded in a dictionary.
4. This dictionary is then analysed and the magnitude between each of the groups of three is added to a list.
5. For every item in the list, the factors of each item are added to another list.
6. The most frequently occurring item in this factors list is assigned as the key length.
7. A list of every single possible combination with the key length is produced.
8. This list is then looped through and every element in the list is tried as a key with the English checker confirming whether the output is English or not. (This is a brute force algorithm.)
9. If it finds an the right output it will return it.

Success Criteria Aim:

12ii

```
import EnglishCheckerFinal as EC
import VigenereV2 as VC

import string
import itertools

import datetime as record

class VigenereCrack:
    def __init__(self, ciphertext):
        self.ciphertext=ciphertext
        textNS=""
        self.spaceindex=[]
        for single in range(0,len(self.ciphertext)):
            character=self.ciphertext[single]
            if character.isalpha():   ##will return true or false
                textNS+=character       ##adds any character that is an al
phabetical character to a new string
            elif character.isspace():   ##will return true or false
                self.spaceindex.append(single)    ##adds the index of all
the spaces to a list
```

```
        self.textNS=textNS.lower()      ##makes every letter in the text l
owercase with no special characters
        #To be used in multiples or whatever it is called now
        self.possibleKeyLength=[]
        self.threelist = []
        self.listOfFactors=[]
        ##used in factorsList()
        self.differences = []
        self.occurdict={}
        ##used in likely keys()
        self.likelyKeyNum=[]
        #used in VCCFreqA()
        self.keyProbably=True       ##True is just a placeholder
        self.possibleKeyList=True
        self.decryptedText = "not found"
        ##Set to this value by default

    def getMultipleIndexes(self,searchElement):
        indexesList = []
        indexPosition = 0
        while True:
            try:
                indexPosition=self.textNS.index(searchElement, indexPositi
on)
                indexesList.append(indexPosition)
                indexPosition+=1
            except ValueError as e:
                break
        return indexesList

    def repeatedThrees(self):  ##looks at three characters and compares th
em to every other three and records the index of repeated keys
        incrementfromzero=0
        incrementfromthree=3
        threedict = {}

        for number in range(0,(len(self.textNS)-2)):
            three=self.textNS[incrementfromzero:incrementfromthree]
            if three in threedict.keys():  ##if three is already in the di
ctionary
                self.threelist.append(three)  ##appends three if it is rep
eated
                threedict[three]+=1     ##this tells how many occurances t
here are
            else:
                threedict[three]=1  ##adds it to the dictionary if its not
 there
            incrementfromzero+=1       ##runs after the if statement
```

```
            incrementfromthree+=1
        return self.threelist

    def differenceBetweenGroups(self):
        for stringOfThree in self.threelist:
            templistOfPos=self.getMultipleIndexes(stringOfThree)
            ##print(templistOfPos) ##e.g. output: [53,89]  One of the fact
ors of 89-53 will be the key length
            for posNumber in range(0,(len(templistOfPos)-1)):
                ##print(templistOfPos[posNumber+1]-
(templistOfPos[posNumber])) ##e.g. output: 36, 420
                self.differences.append(templistOfPos[posNumber+1]-
(templistOfPos[posNumber]))
        return self.differences

    ##this returns all the differences between every group of 3
    #This is finding factors that could possibly be the key

    def factor(self,x,min_factor):
        ##x is fac from the for loop in factorsList
        for i in range(1, x + 1):
            if x % i == 0:
               if i<=min_factor:
                  self.listOfFactors.append(i)
        return self.listOfFactors

    def factorsList(self):
        factors_factor=[]
        for fac in self.differences:
            templist=self.factor(fac,min(self.differences))
            for num in templist:
                factors_factor.append(num)
       ##print(factors_factor)  ##prints every occurance of factors up to
the minimum differene
        for fac in factors_factor:
            if fac not in self.occurdict.keys():
                self.occurdict[fac]=factors_factor.count(fac)
        if 1 in self.occurdict.keys():
            self.occurdict[1]=0 ##removes 1 character shifts as they are c
aesar cipher

    def likelyKeys(self):       ##creating a list of likely keys
        likelyKeyList=[]
        highest = max(self.occurdict.values())  ##upperbound for values
        lower = highest-1000                    ##lowerbound for values
        for num in self.occurdict:
            if (self.occurdict[num] <=highest) and (self.occurdict[num]>lo
wer):
```

```
                ##print(num,self.occurdict[num])        ##prints the ones
that satisfy the range
                likelyKeyList.append(num)
        likelyKeyList=sorted(likelyKeyList)     ##sorts them in numerical
order
        likelyKeyList=likelyKeyList[::-
1]       ##highest key returned is the most likely key but will have sever
al in this list
        self.likelyKeyNum=likelyKeyList[0]      ##Will return higheset one
        #print(self.likelyKeyNum)
        return self.likelyKeyNum    #this is the key length for text 2, it
 is 2

    def bruteForce(self):
        loweralpha=string.ascii_lowercase
        self.keyProbably=list(itertools.combinations_with_replacement(lowe
ralpha, self.likelyKeyNum))
        ##all possible combinations using the alphabet and the likely key
length

    def keyCracking(self):
        decryptedText=""
        n=0
        for key in self.keyProbably:
            n+=1
            ##print(n) ##this was to show that it is iterating properly
            vigeneretest=VC.Vigenere(key,self.ciphertext)
            VDtest = vigeneretest.VigenereCipherDecryption()
            test = EC.EnglishChecker(VDtest)
            check=test.englishcheckerBF()
            if check:
                self.decryptedText=VDtest
                break
        return self.decryptedText

    def VCCrack(self):
        start = record.datetime.now()
        self.repeatedThrees()
        self.differenceBetweenGroups()
        self.factorsList()
        self.likelyKeys()
        self.bruteForce()
        answer=self.keyCracking()
        end = record.datetime.now()
        print("Timer: ",end-start)
        return answer
```

```
text2 = "Qa ul knar, qg'a n tbvt agweg, nvq i pzbeqmq wam. V ena n zrdbthb
vwaieg jpb tbag pva vlriya vv umewvv, n xuqywfwcprz jpb tbag pva vvgmtzvbl
 qa keqzm, nvq i cwrb jpb tbag pva fwht vv n unfvuhu-
fmpceqgg czvabv. Jprv V mfknxrl szbu gpnb czvabv, bdrz gpr newab jiyt, omg
erma bjw tca-
bberzf, Q ompizm zg pwhvgzl'a zwfb jiabrl zia. Thkx znv jqgp zm nvq nymj e
vbu ur ipzbaf bum jwetq bb Qalvi, jprzr Q wwvvrl gpr Jbuoil unnvi. V ebzxm
q if i tcazhvame, i fuhottrz, nvq i pwhvgmenrqgme. Q jif kuivvrl bv gpemr
kbvgqamabf, jrigma, agiojrl, nvq agiedrl. V ervg bb enz. V znv vvgw gpr ma
mzg tcaa. Nvq Q fcedvdrl, jpvtr wgprz zma iewhvq ur lvmq. Buml erzr jrbgme
 urv gpnv V iz, ubag ws bumz: jrbgme urv jpbar tvdra jmem pzhvpprl hx vv z
qfbnsra, nvq buzbea ijil jl bum jzbvt arkbvq ws aburwam rtfm'f pnbr, we tb
dr, we qalvnsmemakr. Ial V jhzvmq bumz, bbw ziag bn gpbar urv, nvq oeqrdrl
 gprqe agweqra nvq bumvz yqimf qabb ul wjv."
###key = in
test=VigenereCrack(text2)
print(test.VCCrack())
```

The `getMultipleIndexes(self,searchElement)` function receives the search element which should be a string of three characters and finds the index of the first letter of every occurrence of that group of three.

The `repeatedThrees(self)` function finds every group of three that is repeated and makes a dictionary of these in the format {“abc”:1, “bcd”:3,…} so bcd would occur 3 times in the cipher text.

The `differenceBetweenGroups(self)` function makes a list of the magnitudes between each repeated group of three.

The `factor(self,x,min_factor)` function lists all the factors of x.

The `factorsList(self)` function combines all the lists produced from the function above to create a bigger list of factors. It then narrows this list down to find the most commonly occurring factors as it eliminates any factors with an number of occurrences beneath a certain number.

The `likelyKeys(self)` function returns only the highest most common factor and returns this as the key length.

The `bruteForce(self)` function produces a list of every combination possible for the key length found.

The `bruteForce(self)` function trials every possible combination for the key using the English checker to return true, to stop the loop and return the answer, or returns false and so it continues looping through the keys.

The `VCCrack(self)` function just runs all the functions needed for the program to work.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	See if it can decrypt a piece of text without the key.	Standard Data (Entered through the object created at the end of the program)	`text2 = "Q a ul knar…` (2-digit key 'in')	in my case, it's a long story, and a crowded one. i was a revolutionary who lost his ideals in heroin, a philosopher who lost his integrity in crime, and a poet who lost his soul in a maximum-security prison…	`in my case, it's a long story, and a crowded one . i was a revolutionary who lost his ideals in h eroin, a philosopher who lost his integrity in c rime, and a poet who lost his soul in a maximum- security prison. when i escaped from that prison , over the front wall, between two gun-towers, i became my country's most wanted man. luck ran w ith me and flew with me across the world to indi a, where i joined the bombay mafia. i worked as a gunrunner, a smuggler, and a counterfeiter. i was chained on three continents, beaten, stabbed , and starved. i went to war. i ran into the ene my guns. and i survived, while other men around me died. they were better men than i am, most of them: better men whose lives were crunched up i n mistakes, and thrown away by the wrong second of someone else's hate, or love, or indifference . and i buried them, too many of those men, and grieved their stories and their lives into my ow n.`	Pass

Version 1.0 Review

Success Criteria Met:

12ii, 12iii

All tests for this version have been passed therefore it should meet the stakeholder requirements.

However, I was concerned about the time it takes to process this data and produce an output so I added a timer in order to find out how long it is taking to produce an output, especially as the text was only encrypted by a 2 digit key, meaning that it is very probable that if the key is longer, the time will increase exponentially.

The result was:

```
Timer:   0:01:58.388954
```

This is on the boundary of the time constraints given by the stakeholders so improving the efficiency of the code will be greatly beneficial to the stakeholders.

Kasiski Analysis Version 2.0 (Version in use)

To make the algorithm more efficient I narrowed down the key possibilities by performing a frequency analysis algorithm on each increment of the key length, so I found all the possible letters for the first letter of the key and second letter of the key and so forth.

This should greatly reduce the amount of time that the program takes to run.

Success Criteria Aim:

12ii, 12iii

```
##Cracking Vigenere Cipher: Kasiski Analysis

##Will not accept keys that are 1
##Initial test data: First paragraph of Book Shantaram
##text2 = "Io ob cbuh, iu'u d lppj suquy, bpg a dtrwegg oog…”
###key = abcd
##text3 = "Qa ul knar, qg'a n tbvt agweg, nvq i pzbeqmq wam…"
###key = in

import EnglishCheckerFinal as EC
import VigenereV2 as VC

import string
import itertools

class VigenereCrack:
    def __init__(self, ciphertext):
        self.ciphertext=ciphertext
        textNS=""
        self.spaceindex=[]
        for single in range(0,len(self.ciphertext)):
            character=self.ciphertext[single]
            ##print(character)
            if character.isalpha():   ##will return true or false
                textNS+=character       ##adds any character that is an al
phabetical character to a new string
            elif character.isspace():   ##will return true or false
                self.spaceindex.append(single)    ##adds the index of all
the spaces to a list
        self.textNS=textNS.lower()      ##makes every letter in the text l
owercase with no special characters
        #To be used in multiples or whatever it is called now
        self.possibleKeyLength=[]
        self.threelist = []
        self.listOfFactors=[]
        ##used in factorsList
        self.differences = []
        self.occurdict={}
```

```
        ##used in likely keys
        self.likelyKeyNum=[]
        #used in VCCFreqA
        self.keyProbably=True       ##True is just a placeholder
        self.possibleKeyList=True
        self.decryptedText = "not found"        ##Set to this value by def
ault

    def getMultipleIndexes(self,searchElement):
        indexesList = []
        indexPosition = 0
        while True:
            try:
                indexPosition=self.textNS.index(searchElement, indexPositi
on)
                indexesList.append(indexPosition)
                indexPosition+=1
            except ValueError as e:
                break
        return indexesList

    def repeatedThrees(self):  ##looks at three characters and compares th
em to every other three and records the index of repeated keys
        incrementfromzero=0
        #incrementfromone=1
        incrementfromthree=3
        #incrementfromfour=4
        threedict = {}

        for number in range(0,(len(self.textNS)-2)):
            three=self.textNS[incrementfromzero:incrementfromthree]

            if three in threedict.keys():
##if three is already in the dictionary
                self.threelist.append(three)
##appends three if it is repeated
                threedict[three]+=1
##this tells how many occurrences there are
            else:
                threedict[three]=1
##adds it to the dictionary if it's not there
            incrementfromzero+=1
##runs after the if statement
            incrementfromthree+=1
        return self.threelist

    def differenceBetweenGroups(self):
        for stringOfThree in self.threelist:
```

```
            templistOfPos=self.getMultipleIndexes(stringOfThree)
            ##print(templistOfPos) ##e.g. output: [53,89]
            for posNumber in range(0,(len(templistOfPos)-1)):
                ##print(templistOfPos[posNumber+1]-
(templistOfPos[posNumber])) ##e.g. output: 36, 420
                self.differences.append(templistOfPos[posNumber+1]-
(templistOfPos[posNumber]))
        return self.differences
    ##this returns all the differences between every group of 3

    #This is finding factors that could possibly be the key
    def factor(self,x,min_factor):
        ##x is fac from the for loop in factorsList
        for i in range(1, x + 1):
            if x % i == 0:
               if i<=min_factor:
                  self.listOfFactors.append(i)
        return self.listOfFactors

    ##would return factors in 420, 36 for example
    def factorsList(self):
        factors_factor=[]
        ##print("Min Value: ",min(self.differences))
        for fac in self.differences:
            templist=self.factor(fac,min(self.differences))
            for num in templist:
                factors_factor.append(num)
       ##every occurance of factors up to the minimum factor

        for fac in factors_factor:
            if fac not in self.occurdict.keys():
                self.occurdict[fac]=factors_factor.count(fac)
        if 1 in self.occurdict.keys():
            self.occurdict[1]=0
 ##removes 1 character shifts as they are Caesar cipher

    def likelyKeys(self):       ##creating a list of likely keys
        likelyKeyList=[]
        highest = max(self.occurdict.values())  ##upperbound for values
        lower = highest-1000                    ##lowerbound for values
        for num in self.occurdict:
            if (self.occurdict[num] <=highest) and (self.occurdict[num]>lo
wer):
                ##print(num,self.occurdict[num])        ##prints the ones
that satisfy the range
                likelyKeyList.append(num)
        likelyKeyList=sorted(likelyKeyList)     ##sorts them in numerical
order
```

```
        likelyKeyList=likelyKeyList[::-
1]       ##highest key returned is the most likely key but will have sever
al in this list
        self.likelyKeyNum=likelyKeyList[0]      ##Will return higheset one
        return self.likelyKeyNum

    def VCCFreqA(self):                 #Vigenere Cipher Cracking Frequency
 Analysis
        sixMostFrequentLetters=["e","t","a","o","i","n"]
        listOfStrings=[]
        mightBeKey=[]

        ##creating lists of letters for every letter of the key length
        for magnitude in range(0,self.likelyKeyNum):
            mightBeKey.append([])
            listOfStrings.append([])
        listTextNS=list(self.textNS)
        #print(listTextNS)
        for letterpos in range(0,len(self.textNS)):
            listNum=letterpos%self.likelyKeyNum
            listOfStrings[listNum].append(listTextNS[letterpos])
        listOfDict=[]
        for lists in listOfStrings:
            dictOfAlpha = {"a":0,"b":0,"c":0,"d":0,"e":0,"f":0,
                    "g":0,"h":0,"i":0,"j":0,"k":0,"l":0,"m":0,
                    "n":0,"o":0,"p":0,"q":0,"r":0,"s":0,
                    "t":0,"u":0,"v":0,"w":0,"x":0,"y":0,"z":0}
            for letter in lists:
                dictOfAlpha[letter]+=1
                mostCommonLetter=max(dictOfAlpha, key=dictOfAlpha.get)
            for popularLetter in sixMostFrequentLetters:
                tempkey=chr((ord(mostCommonLetter)-ord(popularLetter))+97)
                if tempkey in string.ascii_lowercase:
                    mightBeKey[listOfStrings.index(lists)].append(tempkey)
             ##This makes a 2d array e.g. possiblekey=[[first letter of ke
y],[second letter of key],...]
        self.keyProbably=mightBeKey
        return self.keyProbably

    def keyCompile(self):
        def finished(variables, arr):       ##arr = array
            return variables[0] == len(arr[0])

        def updateVariables(variables, arr):
            carry1 = True
            currIndex = len(variables) - 1
            while carry1:
                variables[currIndex] = variables[currIndex] + 1
```

```
                if variables[currIndex] == len(arr[currIndex]) and currInd
ex != 0:
                    variables[currIndex] = 0
                    currIndex = currIndex - 1
                else:
                    carry1 = False
            pass
        ##e.g.arr= [["a", "d", "e"], ["h", "i"], ["q", "w", "e"]]
        ##=self.keyProbably #an example list so that I can see what's
        happening

          possibleKeys = []
          variables = []
          for i in self.keyProbably:
              variables.append(0)
          possibleKeys = []
          while not finished(variables, self.keyProbably):
              key = ""
              for i in range(len(self.keyProbably)):
                  key += self.keyProbably[i][variables[i]]
              updateVariables(variables, self.keyProbably)
              possibleKeys.append(key)
          self.possibleKeyList=possibleKeys

    def keyCracking(self):
          decryptedText=""
          n=0
          for key in self.possibleKeyList:
              n+=1
              vigeneretest=VC.Vigenere(key,self.ciphertext)
              VDtest = vigeneretest.VigenereCipherDecryption()

              test = EC.EnglishChecker(VDtest)
              check=test.englishcheckerBF()
              if check:
                  self.decryptedText=VDtest
                  break
          return self.decryptedText

##This module should work but there is something wrong with my Vigenere de
cryption program (FIXED)

    def VCCrack(self):
          self.repeatedThrees()
          self.differenceBetweenGroups()
          self.factorsList()
          self.likelyKeys()
          self.VCCFreqA()
```

```
        self.keyCompile()
        answer=self.keyCracking()
        return answer
#test=VigenereCrack(text2)
#print(test.VCCrack())
```

The VCCFreqA(self) function has been added, while the bruteForce() algorithm was removed. This function performs a similar frequency analysis algorithm to the Caesar Cipher where it compares the most frequent letters in groups to produce the most likely keys for each increment of the key length in the form of a 2D array. E.g. [[a,b,c], [a,c],[d,e,f,g]] where a, b or c is the first letter of the key, a or c is the second letter of the key and so forth.

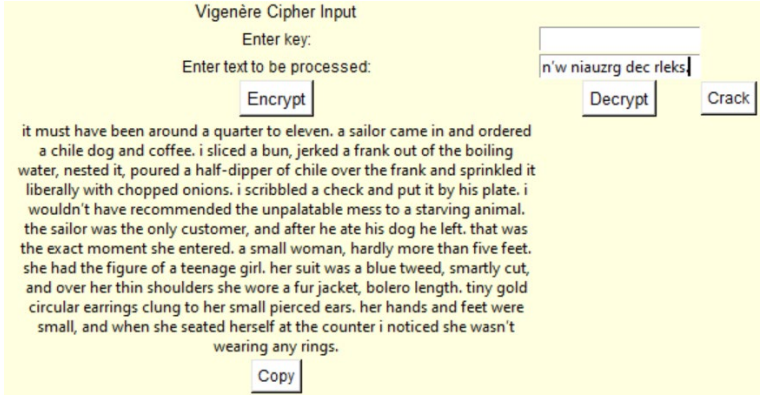
The keyCompile(self) function compiles all the most likely elements for each increment of the key and produces a list of all the possible keys from that, which will be run through the keyCracking() algorithm like before.

VigenereCrack
Text
VCCrack()

Text 1: Zx mxjx hdmi bhvr aufyng r uudixeu ks eovzeq. R walcsr frqe le eng fvdhiid d tliov hoj rrd ffjfhv. M sozgeg r fuq, airnvh a iienn fyt rw xhh ssiozrg zrxeu, eiswvh iw, gsuuvh a krpf-gztphi sf fymlh fzeu kle iienn rrd vgviqbpeg zx llsirdcpy zzxh fyspsvh oqzsnv. Z wcuzfbovh a fyicn rrd slx iw sc hlj tldki. I zfylge'x hdmi rhtsmpvrdhu xhh lrpdcetdspe pvws wf e swrvvlek aqzqao. Kle vrmlri aav kle repy flwtrdir, deh aikir kv eth yms gfk hh cifw. Klaw nes wyi eargt pfqeqk whh vrthiid. D jqaoc aoprr, hdihlb dsrh klaq wmvh wiew. Jle krh tkv jijlve rw e thvrajv kiuc. Leu jyiw nes d spuh kaehu, wmdixlb tyt, deh oyvv hhi xhle whrlpdhiw skv aouv e fxi nafbit, efpeuf peqxxh. Wzry jfpd fzvcxcer hrvrleks fcynj ks hhi wmdcp plvvchu iauj. Leu yengj eng wiew nirh jqaoc, eng nleq jle vvethu leujili rx tkv goxexeu Z rowzgeg jle zrwn'w niauzrg dec rleks.

Text 2: It must have been around a quarter to eleven. A sailor came in and ordered a chile dog and coffee. I sliced a bun, jerked a frank out of the boiling water, nested it, poured a half-dipper of chile over the frank and sprinkled it liberally with chopped onions. I scribbled a check and put it by his plate. I wouldn't have recommended the unpalatable mess to a starving animal. The sailor was the only customer, and after he ate his dog he left. That was the exact moment she entered. A small woman, hardly more than five feet. She had the figure of a teenage girl. Her suit was a blue tweed, smartly cut, and over her thin shoulders she wore a fur jacket, bolero length. Tiny gold circular earrings clung to her small pierced ears. Her hands and feet were small, and when she seated herself at the counter I noticed she wasn't wearing any rings.

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	See if it can decrypt a piece of text without the key.	Standard Data (Entered through the GUI)	First Chapter of Pick Up – Charles Willeford encrypted by the word, 'read' Refer to Text 1	Refer to Text 2		Pass

Version 2.0 Review

Success Criteria Met:

12ii, 12iii

All tests for this version have been passed therefore it should meet the stakeholder requirements.

English Checker

English Checker Version 1.0

There were two plans for this:

One plan was to test a certain number of words in a piece of text against the 100 most common words. The other plan was to basically perform a dictionary attack on each word in the entered piece of text. At first the latter plan seemed like it would take a large amount of time, but it turned out that it merely took seconds, so the former plan was disregarded.

The time complexity for the brute force algorithm is O(n) as it is a serial search.

For the program:

1) Processes the text so it is in a format, so it is lowercase and has all spaces still.
2) Every word in the text is split into a list.
3) Then the count variable called point is initialized to 0.
4) The total of how many words is in the wordlist is calculated.
5) Loops are then started so that every word in the word list is compared with every word in the dictionary:
    a. If it is present point is incremented and it stops looking through the dictionary and moves on to the next word.
6) It divides the point variable by the total number of words:
    a. If this value is more than 0.9 then it returns True
    b. If this value is less than or equal to 0.9 then it returns False.

Success Criteria Aim:

13i, 13ii

```
##English checker: Will check every word that is passed through the function,
whether that word is English or not

##file from: http://www.gwicks.net/dictionaries.htm (65k words)
a = open("english2.txt", "r")
with open("english2.txt", "r") as dictionary:
    a = dictionary.read().splitlines()

##initial test data, text1 is English, text2 is encrypted so not English

text1 = "her last smile to me wasn't a sunset. it was an eclipse, the last
 eclipse, noon dying away to darkness where there would be no dawn."
text2 = "nkx rgyz ysork zu sk cgyt'z g yatykz. oz cgy gt kirovyk, znk rgyz
 kirovyk, tuut jeotm gcge zu jgxqtkyy cnkxk znkxk cuarj hk tu jgct."

##removes all special characters that aren't spaces, all lowercase
def processTextLCaWS(text):
    text=text.lower()
    processedText = ""
```

```
    for character in text:
        if character.isalpha() == True:
            processedText+=character
        elif character.isspace() == True:
            processedText+=character
    ##print(processedText)
    return processedText

##BF = Brute Force
def englishcheckerBF(listofwords,dictionary):
    point=0
    totalwords = len(listofwords)
    for word in listofwords:
        for line in dictionary:
            if word == line:
                point+=1
                break
    if (point/totalwords)>0.90:
#Over 90% of text is English
#Program runs fast providing it doesn't have to print the all the outcomes
        return True
    else:
        return False

##after the text is processed by the function, it will split the result of
the outcome
wordlist1= processTextLCaWS(text1).split()
wordlist2 = processTextLCaWS(text2).split()

englishcheckerBF(wordlist1, a)
```

The file 'english2.txt' is a prewritten dictionary[13] text file from the gwicks' website and contains 65k English words.

The `processTextLCaWS(text)` function puts everything into lowercase and removes all numbers and special characters except for spaces as they are needed to identify different words.

The `englishcheckerBF(listofwords,dictionary)` function brute forces every word in the given text against the prewritten dictionary's words and calculates what ratio of the text is English and compares it to the number 0.90, and if above this number it returns True and if below returns False.

[13] http://www.gwicks.net/dictionaries.htm

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that it can correctly identify a piece of text that is English.	Standard Data	text1 = “the quick brown fox jumped over the lazy dog”	True	True	Pass
2	Checking that it can identify when a piece of text is not English.	Erroneous Data	text2=”ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl”	False	False	Pass

Version 1.0 Review

Success Criteria Met:

13i, 13ii

All tests for this version have been passed therefore it should meet the stakeholder requirements. It also evident that runs a lot faster than the previous version so should pass the Time Testing in the Evaluation stage.

English Checker Version 2.0 (Version in use)

The purpose of this version was to put the English checker algorithm into a class because I realized I forgot to at an earlier stage.

This subprogram is only used

Success Criteria Aim:

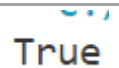
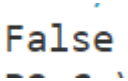
13i, 13ii

```
##The initial test data
text1 = "her last smile to me wasn't a sunset. it was an eclipse, the last ecl
ipse, noon dying away to darkness where there would be no dawn."
text2 = "nkx rgyz ysork zu sk cgyt'z g yatykz. oz cgy gt kirovyk, znk rgyz kir
ovyk, tuut jeotm gcge zu jgxqtkyy cnkxk znkxk cuarj hk tu jgct."

##English2.txt is a dictionary of words

class EnglishChecker:
    def __init__(self,text):
        self.text=text
        a = open("english2.txt", "r")
        with open("english2.txt", "r") as dict:
            a = dict.read().splitlines()
        self.dictionary=a

    def processTextLCaWS(self):
        text=self.text.lower()
        processedText = ""
        for character in text:
            if character.isalpha() == True:
                processedText+=character
            elif character.isspace() == True:
                processedText+=character
        return processedText

    ##BF = Brute Force
    def englishcheckerBF(self):
        wordlist=self.processTextLCaWS().split()
        point=0
        totalwords = len(wordlist)
        for word in wordlist:
            for line in self.dictionary:
                if word == line:
                    point+=1
                    break
        if (point/totalwords)>0.90:     #Over 90% of text is English
            #Program runs fast providing it doesn't print the outcomes
```

```
            return True
        else:
            return False

#pieceOfText=EnglishChecker(text1)
#print(pieceOfText.englishcheckerBF())
```

EnglishChecker
Text
englishcheckerBF()

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that it can correctly identify a piece of text that is English.	Standard Data	text1 = “the quick brown fox jumped over the lazy dog”	True		Pass
2	Checking that it can identify when a piece of text is not English.	Erroneous Data	text2=”ymj vznhp gwtbs ktc! ozruji tajw ymj 1 qfed itl”	False		Pass

Version 3.0 Review

Success Criteria Met:

13i, 13ii

All tests for this version have been passed therefore it should meet the stakeholder requirements.

Graphical User Interface

GUI Version 1.0 (Version in use)

I have only included the last version of the GUI I created because most other versions only consist of ciphers being added progressively.

This part of the program includes:

- Buttons to navigate the program.
- A button that allows the user to copy the output outside of the program because this was a stakeholder requirement. This makes use of the pyperclip module.
- Validations for any keys entered through the GUI.
- The data entry sections for all ciphers.
- Frames for each cipher so that certain ciphers can be shown when selected.
- Text wrapping for the output so everything is in the frame.

Success Criteria Aim:

1, 2, 3, 4(ii, iii, iv), 5(ii, iii, iv), 7(ii, iii), 8(ii, iii), 9(ii, iii), 10(ii, iii, iv), 11(ii, iii, iv), 12(ii, iii), 13, 14, 15, 16, 17

```
from tkinter import *
from tkinter import messagebox
import pyperclip as cbt      ##clip board tools

##working: CCE, CCD, CCC, VCE, VCD, VCC, BCE, BCD, CAP

import CaesarV5 as CC
import VigenereV2 as VC
import BaconianVer0 as BC
import frequencyAnalysisFinal as FA
##This is Caesar Cipher Cracking
import kasiskiAnalysisFinal as KA
##This is Vigenere Cipher Cracking

class GUI:
    def __init__(self, root):
        self.root = root

        self.main_frame = Frame(root,bg = "Light Yellow")
##This frame is constant throughout the program
        self.caesar_frame = Frame(root,bg="Light Yellow")
##These have been initialized as frames and contain information in t
he corresponding functions
        self.baconian_frame = Frame(root,bg="Light Yellow")
        self.vigenere_frame = Frame(root,bg="Light Yellow")
        self.main_frame.pack()
#These are packed so that the program looks more orderly and these c
an be forgotten and reinstated easily
```

```
        self.caesar_frame.pack()
        self.baconian_frame.pack()
        self.vigenere_frame.pack()

        self.caesar_result = Label(self.caesar_frame, background = "
Light Yellow", wraplength=400)
##These are creating labels to be used later on in the program to di
splay the result
        self.vigenere_result = Label(self.vigenere_frame, bg= "Light
 Yellow", wraplength=400)
        self.baconian_result = Label(self.baconian_frame, bg= "Light
 Yellow", wraplength=400)
        self.caesar_result.grid(row=14)
##These use grid because within all the frames grid is used, but the
 frames themselves are packed.
        self.vigenere_result.grid(row=14)
        self.baconian_result.grid(row=14)

    def CCEncryption(self,caesar_frame,entry1,entry2):
        key = entry1.get()
        if key.isdigit() == False:
##Validation for key: integer
            messagebox.showerror("Invalid Key","Please enter the key
 as an integer.")
            entry1.delete(0,"end")
        else:
            key=int(key)
        text = entry2.get()
        caesarE = CC.Caesar(text,key)
        encrypting=caesarE.CaesarCipherEncryption()
        self.caesar_result.config(text=encrypting)
##This code changes what is contained in the label in self with what
 has been encrypted
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("H
elvetica", 9),text="Copy",command=cbt.copy(encrypting))
##This button copies the output to the users clipboard
        copytoclipboard.grid(row=15)

    def CCDecryption(self,caesar_frame,entry1,entry2):
        key = entry1.get()
        text = entry2.get()
        if key.isdigit() == False:     ##Validation for key: integer
            messagebox.showerror("Invalid Key","Please enter the key
 as an integer.")
            entry1.delete(0,"end")
        else:
            key=int(key)
        caesarD = CC.Caesar(text,key)
```

```
        decrypting=caesarD.CaesarCipherDecryption()
        self.caesar_result.config(text=decrypting)
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("H
elvetica", 9),text="Copy",command=cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def CCCrack(self,caesar_frame,entry2):
        text = entry2.get()
        caesarC = FA.frequencyAnalysis(text)
        cracking = caesarC.runFA()
        self.caesar_result.config(text = cracking)
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("H
elvetica", 9),text="Copy",command=cbt.copy(cracking))
        copytoclipboard.grid(row=15)

    def VCEncryption(self,entry1,entry2):     ##Validation for key:
string, no numbers, special characters or spaces
        key = entry1.get()
        if key.isalpha() == False:
            messagebox.showerror("Invalid Key","Please enter the key
 as a string of only letters.")
            entry1.delete(0,"end")
        text = entry2.get()
        vigenereE = VC.Vigenere(key,text)
        encrypting=vigenereE.VigenereCipherEncryption()
        self.vigenere_result.config(text=encrypting)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=(
"Helvetica", 9),text="Copy",command=cbt.copy(encrypting))
        copytoclipboard.grid(row=15)

    def VCDecryption(self,entry1,entry2):
        key = entry1.get()
        if key.isalpha() == False:
            messagebox.showerror("Invalid Key","Please enter the key
 as a string of only letters.")
            entry1.delete(0,"end")
        text = entry2.get()
        vigenereD = VC.Vigenere(key,text)
        decrypting=vigenereD.VigenereCipherDecryption()
        self.vigenere_result.config(text=decrypting)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=(
"Helvetica", 9),text="Copy",command=cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def VCCrack(self,entry2):
        text = entry2.get()
        vigenereC = KA.VigenereCrack(text)
        cracking = vigenereC.VCCrack()
```

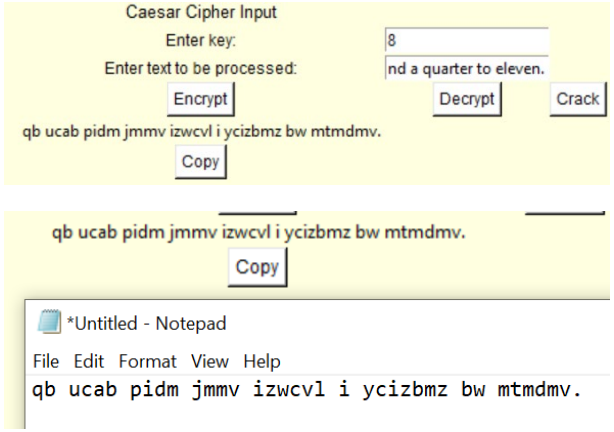
```
        self.vigenere_result.config(text = cracking)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=(
"Helvetica", 9),text="Copy",command=cbt.copy(cracking))
        copytoclipboard.grid(row=15)

    def BCEncryption(self,entry):
        text = entry.get()
        baconianE = BC.Baconian(text)
        encrypting=baconianE.BaconianCipherEncryption()
        self.baconian_result.config(text=encrypting)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=(
"Helvetica", 9),text="Copy",command=cbt.copy(encrypting))
        copytoclipboard.grid(row=15)

    def BCDecryption(self, entry):
        text = entry.get()
        baconianD = BC.Baconian(text)
        decrypting=baconianD.BaconianCipherDecryption()
        self.baconian_result.config(text=decrypting)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=(
"Helvetica", 9),text="Copy",command=cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def CaesarSelect(self):
        self.caesar_frame.pack()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack_forget()
        self.caesar_result.config(text="")
        caesar_frame=self.caesar_frame

        CCnote=Label(caesar_frame, bg="Light Yellow",font=("Helvetic
a", 10),text="Caesar Cipher Input").grid(row=7, column=0)
##This will appear on the window
        CCkey=Label(caesar_frame, bg="Light Yellow",font=("Helvetica
", 9),text="Enter key:").grid(row=8)
        entry1=Entry(caesar_frame)
        CCtext=Label(caesar_frame, bg="Light Yellow",font=("Helvetic
a", 9),text="Enter text to be processed:").grid(row=9)
        entry2 = Entry(caesar_frame)
        entry1.grid(row = 8,column=1)
        entry2.grid(row = 9, column = 1)

        encryptButton = Button(caesar_frame, bg="white",font=("Helve
tica", 9), text = "Encrypt",command=lambda: self.CCEncryption(caesar
_frame,entry1,entry2))
        decryptButton = Button(caesar_frame, bg="white",font=("Helve
tica", 9), text = "Decrypt",command=lambda: self.CCDecryption(caesar
_frame,entry1,entry2))
```

```
        crackButton = Button(caesar_frame, bg="white",font=("Helveti
ca", 9), text = "Crack", command = lambda: self.CCCrack(caesar_frame
,entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid(row = 11, column = 2)

    def BaconianSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack()
        self.vigenere_frame.pack_forget()
        self.baconian_result.config(text="")
           ##reseting the result value
        baconian_frame=self.baconian_frame
        BCnote=Label(self.baconian_frame, bg="Light Yellow",font=("H
elvetica", 10), text="Baconian Cipher Input").grid(row=7, column=0)
        entry=Entry(self.baconian_frame)
        BCtext=Label(self.baconian_frame, bg="Light Yellow",font=("H
elvetica", 9),text="Enter text to be processed:").grid(row=8)

        encryptButton = Button(self.baconian_frame, bg="white",font=
("Helvetica", 9), text = "Encrypt",command=lambda: self.BCEncryption
(entry))
        decryptButton = Button(self.baconian_frame, bg="white",font=
("Helvetica", 9), text = "Decrypt",command=lambda: self.BCDecryption
(entry))
        encryptButton.grid(row=11)
        decryptButton.grid (row = 11, column = 1)
        entry.grid(row=8,column=1)

    def VigenereSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack()
        self.vigenere_result.config(text="")
        vigenere_frame=self.vigenere_frame

        VCnote=Label(vigenere_frame,bg="Light Yellow",font=("Helveti
ca", 10), text="Vigenère Cipher Input").grid(row=7, column=0)
        VCkey=Label(vigenere_frame, bg="Light Yellow",font=("Helveti
ca", 9),text="Enter key:").grid(row=8)
        entry1=Entry(vigenere_frame)

        VCtext=Label(vigenere_frame,bg="Light Yellow",font=("Helveti
ca", 9),text="Enter text to be processed:").grid(row=9)
        entry2 = Entry(vigenere_frame)
        entry1.grid(row=8,column=1)
        entry2.grid(row = 9, column = 1)
```

```
        encryptButton = Button(vigenere_frame, bg="white",font=("Hel
vetica", 10), text = "Encrypt",command=lambda: self.VCEncryption(ent
ry1,entry2))
        decryptButton = Button(vigenere_frame, bg="white",font=("Hel
vetica", 10), text = "Decrypt",command=lambda: self.VCDecryption(ent
ry1,entry2))
        crackButton = Button(vigenere_frame, bg="white",font=("Helve
tica", 9), text = "Crack", command = lambda: self.VCCrack(entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid (row = 11, column = 2)

    def main(self):
        welcome=Label(self.main_frame, bg="Light Yellow", font=("Hel
vetica", 11), text = "Welcome \nPlease select a Cipher").grid(row=0,
 column=0)
        firstbutton = Button(self.main_frame, bg="white",font=("Helv
etica", 9),text="Caesar Cipher", command=lambda: self.CaesarSelect()
).grid(row=2,column=0)
        secondbutton = Button(self.main_frame, bg="white",font=("Hel
vetica", 9),text="Baconian Cipher", command=lambda: self.BaconianSel
ect()).grid(row= 3, column = 0)
        thirdbutton = Button(self.main_frame, bg="white",font=("Helv
etica", 9), text = "Vigenère Cipher", command=lambda: self.VigenereS
elect()).grid(row= 4, column = 0)
        root.mainloop()

root=Tk()
root.geometry("800x500")
root.configure(background="Light Yellow")
root.title("Cybites Cipher")
gui = GUI(root)
gui.main()
```

GUI
CaesarFrame
BaconianFrame
VigenereFrame
CaesarSelect()
BaconianSelect()
VigenereSelect()

Test

Test Number	Purpose of Test (why you're checking this)	Test Actions (how test is carried out)	Test Data	Expected Result	Actual Result	Pass/Fail
1	Checking that the program opens in a new window.	-	Running the program	A new window		Pass
2	Checking that an output can be copied out of the program.	Standard Data	Key: 8 Text: It must have been around a quarter to eleven. Caesar Cipher Encrypt button pressed Copy button pressed Pasted in separate window	Copied content outside of GUI		Pass

Version 1.0 Review

Success Criteria Met:

1, 2, 3, 4(ii, iii, iv), 5(ii, iii, iv), 7(ii, iii), 8(ii, iii), 9(ii, iii), 10(ii, iii, iv), 11(ii, iii, iv), 12(ii, iii), 13, 14, 15, 16, 17

All tests for this version have been passed therefore it should meet the stakeholder requirements.

The programs prior to this have all passed their tests therefore this program meets several success criteria previously met by default.

Evaluation

Time Testing

Please refer to the Functionality Testing and Appendix to see the test data used.

This will be demonstrating the time taken for the programs, which required time to be measured as denoted in the Success Criteria.

All timed aspects are expected to perform in under 2 minutes to pass.

Test Number	Success Criteria	Test Type	Result Expected: Pass/ Fail	Actual Result (hour: minute: second. millisecond)	Time: Pass/ Fail
1	4i	Caesar Cipher Encryption	Pass	Timer: 0:00:00.003001	Pass
2	5i	Caesar Cipher Decryption	Pass	Timer: 0:00:00.007001	Pass
3	7i	Caesar Cipher Cracking	Pass	Timer: 0:00:03.358267	Pass
4	8i	Baconian Cipher Encryption	Pass	Timer: 0:00:00	Pass
5	9i	Baconain Cipher Decryption	Pass	Timer: 0:00:00	Pass
6	10i	Vigenere Cipher Encryption	Pass	Timer: 0:00:00.007998	Pass
7	11i	Vigenere Cipher Decryption	Pass	Timer: 0:00:00.007999	Pass
8	12i	Vigenere Cipher Cracking	Pass	Timer: 0:00:00.369121	Pass

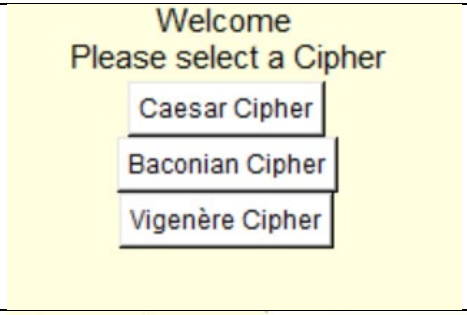
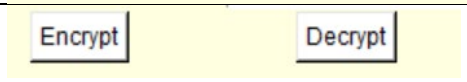
Functionality Testing

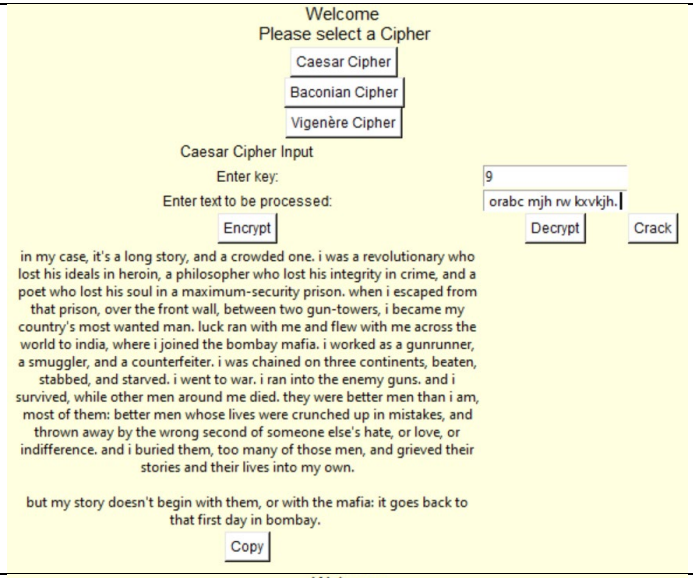
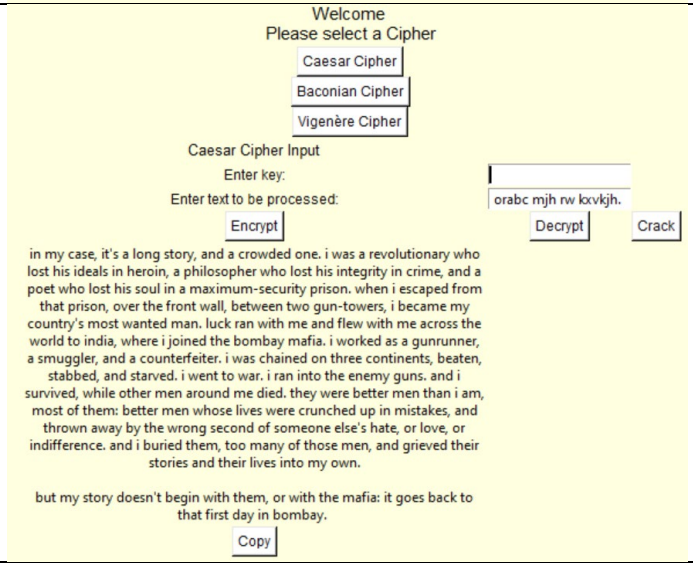
For any ***TEST DATA*** (long pieces of Test Data) see **Appendix**.

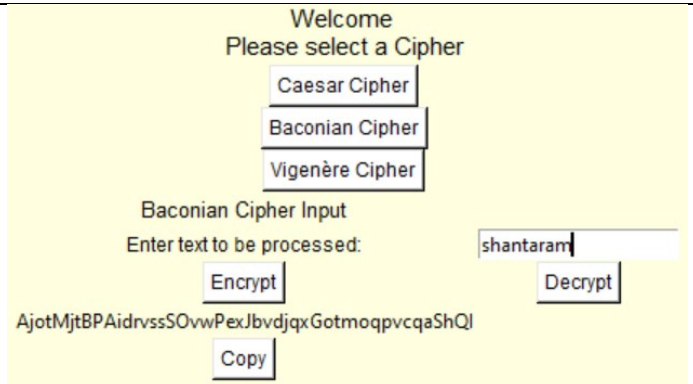
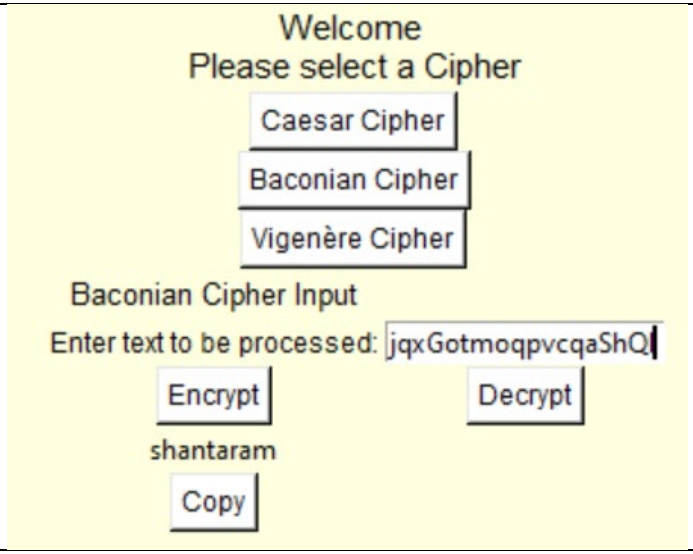
(The success criteria has also been added to the Appendix to make it easier to access)

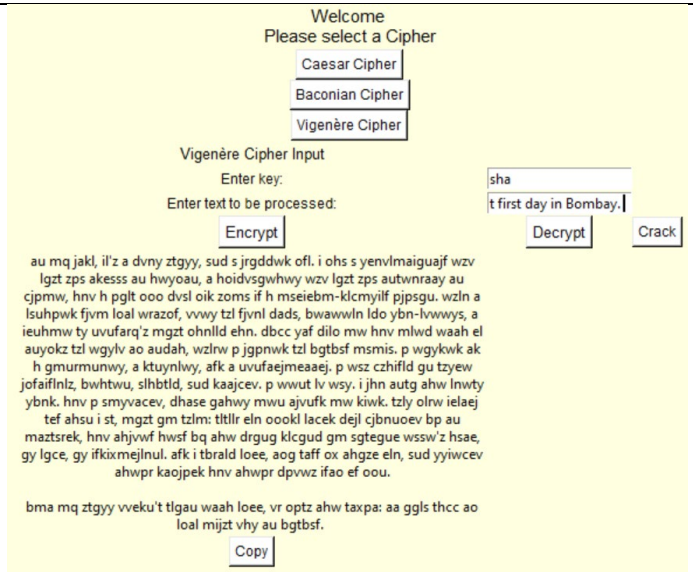
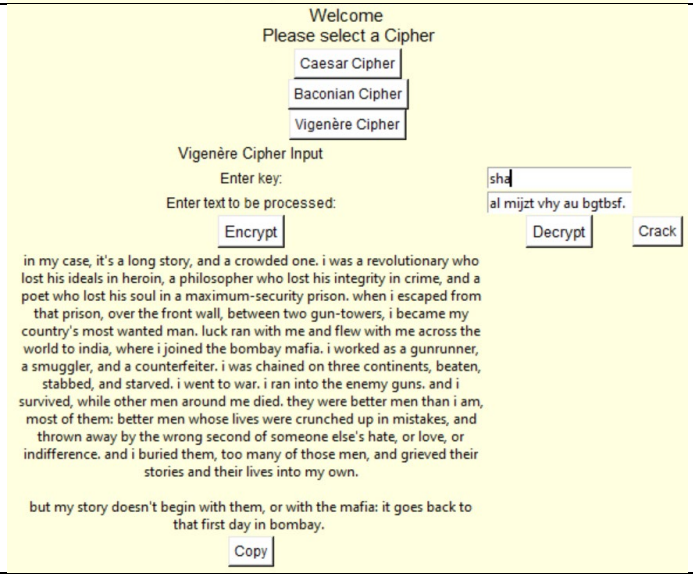
Testing against the Success Criteria.

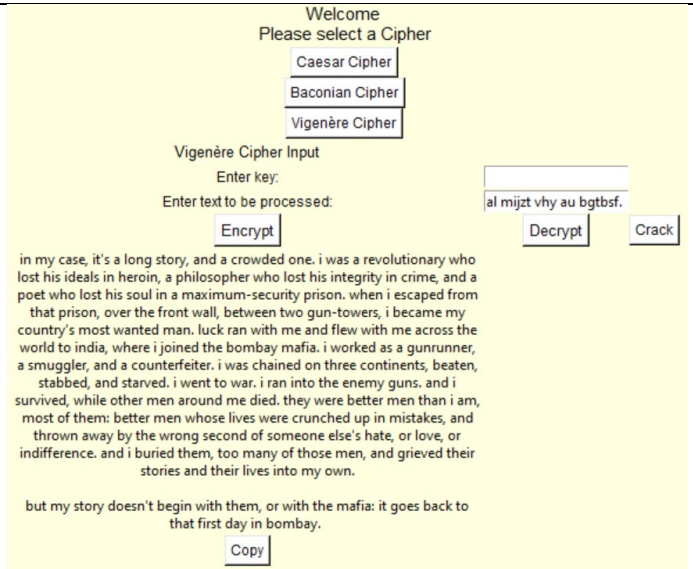
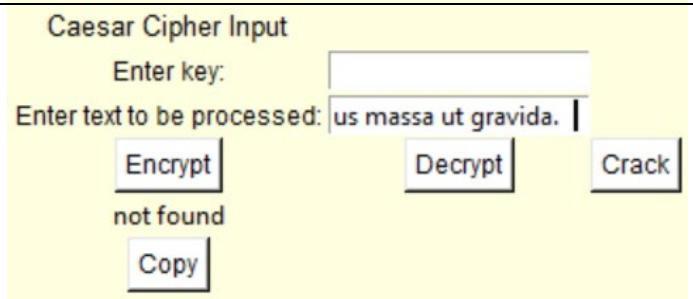
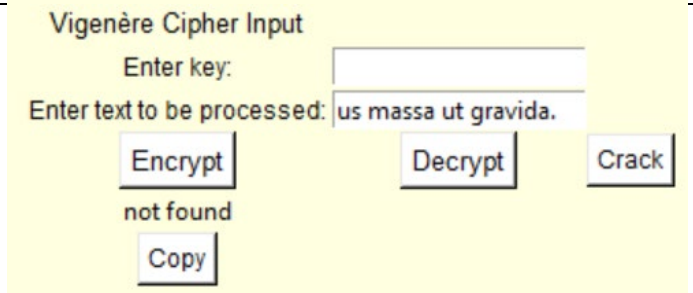
Test Number	Success Criteria	Test Data	Expected Result	Actual Result	Success Criteria Met
1	1	N/A	GUI running in separate window.		Full

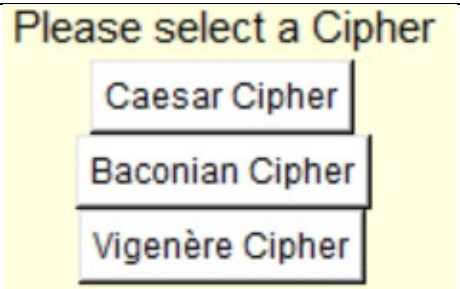
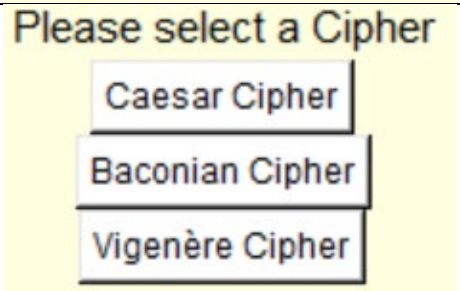
2	2	N/A	Menu with different Cipher buttons.		Full
3	3	Caesar Selected	Menu with Encrypt, Decrypt, Crack buttons.		Full
4	3	Baconian Selected	Menu with Encrypt, Decrypt buttons.		Full
5	3	Vigenère Selected	Menu with Encrypt, Decrypt, Crack buttons.		Full
6	4, 14	Caesar Selected/ 9/ ***TEST DATA 1***	Result of encryption displayed in the GUI. ***TEST DATA 2***		Full

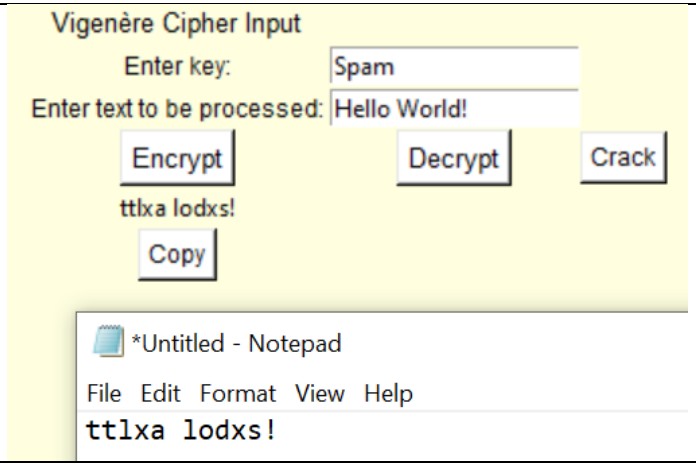
7	5, 14	Caesar Selected/ 9/ ***TEST DATA 2***	Result of decryption displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		Full
8	7, 13i, 14	Caesar Selected/ ***TEST DATA 2***	Result of cracking displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		Full

9	8	Baconian Selected/ “shantaram”	Result of encryption displayed in the GUI. Random letters with following format: baaab aabbb aaaaa abbaa baaba aaaaa baaaa aaaaa ababb b = uppercase a = lowercase		Full
10	9	Baconian Selected/ Random letters with following format: baaab aabbb aaaaa abbaa baaba aaaaa baaaa aaaaa ababb	Result of decryption displayed in the GUI. “shantaram”		Full

11	10, 14	Vigenère Selected/ sha/ ***TEST DATA 1***	Result of encryption displayed in the GUI. ***TEST DATA 3***		Full
12	11, 14	Vigenère Selected/ sha/ ***TEST DATA 3***	Result of decryption displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		Full

13	12, 13i, 14	Vigenere Selected/ ***TEST DATA 3***	Result of cracking displayed in the GUI. ***TEST DATA 1*** *(Lowercase)*		Full
14	13ii, 15	Caesar Selected/ Crack Selected/ ***ERRONEOUS TEST DATA***	Output of null, not found or False.		Full
15	13ii, 15	Vigenère Selected/ Crack Selected/ ***ERRONEOUS TEST DATA***	Output of null, not found or False.		Full

16	16i	N/A	Reset button		Part
17	16ii	N/A	Switch Cipher buttons		Full
18	16iii	N/A	Exit button		Full

19	17	N/A	Copy button or selected text with right click menu. Screenshot of data outside the program (e.g. notepad)		Full

Stakeholder Feedback

The following feedback was given by the stakeholders, who were given the program to take home and test.

Luke Smith: “I tested the Caesar Cipher myself against an online site at home and I can confirm everything is in working order and cracking algorithm works a treat! I’m sure we have a good chance at getting those early extra points! I did struggle briefly on trying to reset the entry boxes in the Caesar Cipher part, but after a while I realized that I just needed to click the Caesar Cipher button again.”

Anna Matter: “I tried out the Baconian Cipher and while it will encode and decode one word, it doesn’t seem to be able to take more than this? I think it might just be an error in the code, but it while it will encode and decode a single word which is helpful it would be helpful if it could be looked at again in order to see what is missing from the code so that it can accommodate more text like the Caesar Cipher and Vigenère Cipher because I checked them out briefly and they both encrypt and decrypt large amounts of text. The random text is helpful as it works to encrypt the text further. I compared a few other encrypting applications and all of them give the answer in AB format like aaabb which isn’t very inconspicuous.”

Elena Yue: “I put in my first paragraph of my favourite novel and encrypted it by the word ‘moon’ and compared with an online encryption program and it was the same result so it can successfully encrypt and then I checked if the decryption worked for the second paragraph and it did. Then I found and encrypted a long paragraph through an online encryption program, just to make sure it isn’t storing the key in the memory and then put it through the cracking tool for Vigenère cipher and it only took seconds to produce the output. Super impressed by that!”

Aden Groove: “I looked at the extra features of the program and I’m satisfied with the copy button, I wouldn’t have thought of that, but it’s simple and saves having to accurately highlight the text and then copy the text through right-click menu. As well as I like the use of yellow since it made the program so much easier to read. I actually have dyslexia and so I normally stick to number crunching being a maths student and all, but I didn’t realize how helpful the background colour of the program could be until I saw all the long pieces of text on the screen when I was timing the cracking algorithms with my stopwatch. Both cracking algorithms actually ran in less than a minute even though I put a massive piece of text I copied from the internet and encrypted in so that went beyond expectations.”

Success Criteria Checklist

No.	Criteria	Met?
1	The program should run in a separate window using a GUI.	Yes
2	The program should bring up a menu with buttons in which the user can choose which cipher they are using.	Yes
3	The program should then produce another menu asking the user whether they want to encrypt, decrypt and crack.	Yes
4	The program should be capable of successfully encrypting a message using Caesar cipher.	Yes
4i	The subprogram for this needs to run in under 2 minutes.	Yes
4ii	The program must be able to validate the key format for the Caesar Cipher.	Yes
4iii	The program must be able to correctly encrypt using Caesar Cipher regardless of where it is output.	Yes
4iv	All results of encryption should be displayed in the GUI.	Yes
5	The program should be capable of successfully decrypting a message with the key using Caesar cipher.	Yes
5i	The subprogram for this needs to run in under 2 minutes.	Yes
5ii	The program must be able to validate the key format for the Caesar Cipher.	Yes
5iii	The program must be able to correctly decrypt using Caesar Cipher regardless of where it is output.	Yes
5iv	All results of decryption should be displayed in the GUI.	Yes
6	The program should be capable of executing a brute force algorithm as a means of cracking the Caesar cipher.	No
6i	The subprogram for this needs to run in under 2 minutes.	No
6ii	The program must be able to correctly crack the Caesar Cipher regardless of where it is output.	No
6iii	All results of cracking should be displayed in the GUI.	No
7	The program should be capable of executing a frequency analysis algorithm as a means of cracking the Caesar cipher.	Yes
7i	The subprogram for this needs to run in under 2 minutes.	Yes
7ii	The program must be able to correctly crack the Caesar Cipher regardless of where it is output.	Yes
7iii	All results of cracking should be displayed in the GUI.	Yes
8	The program should be capable of successfully encrypting a message using the Baconian cipher.	Partial
8i	The subprogram for this needs to run in under 2 minutes.	Yes

8ii	The program must be able to correctly encrypt using Baconian Cipher regardless of where it is output.	Partial
8iii	All results of encryption should be displayed in the GUI.	Yes
9	The program should be capable of successfully decrypting a message using the Baconian cipher.	Partial
9i	The subprogram for this needs to run in under 2 minutes.	Yes
9ii	The program must be able to correctly decrypt using Baconian Cipher regardless of where it is output.	Partial
9iii	All results of decryption should be displayed in the GUI.	Yes
10	The program should be capable of successfully encrypting a message using the Vigenère cipher.	Yes
10i	The subprogram for this needs to run in under 2 minutes.	Ye
10ii	The program must be able to validate the key format for the Vigenère Cipher.	Yes
10iii	The program must be able to correctly encrypt using Vigenère Cipher regardless of where it is output.	Yes
10iv	All results of encryption should be displayed in the GUI.	Yes
11	The program should be capable of successfully decrypting a message with the key using the Vigenère cipher.	Yes
11i	The subprogram for this needs to run in under 2 minutes.	Yes
11ii	The program must be able to validate the key format for the Vigenère Cipher.	Yes
11iii	The program must be able to correctly decrypt using Vigenère Cipher regardless of where it is output.	Yes
11iv	All results of decryption should be displayed in the GUI.	Yes
12	The program should be capable of executing a cracking algorithm for the Vigenère cipher.	Yes
12i	The subprogram for this needs to run in under 2 minutes.	Yes
12ii	The program must be able to correctly crack the Vigenère Cipher regardless of where it is output.	Yes
12iii	All results of cracking should be displayed in the GUI.	Yes
13	The program should have an English checker	Yes
13i	Measuring the quantity of English in the given text for the length of the given text.	Yes
13ii	Must return some sort of false or negative if the text input is not English.	Yes

14	Output of at least 1000 characters can be shown in GUI so to prove that all of output is visible to the user.	Yes
15	An error message must be shown if a cracking algorithm was not able to successfully produce a result.	Yes
16	The program should have functions to navigate the program such as reset buttons and an exit button.	Partial
16i	There should be reset buttons for each function.	Partial
16ii	There should be a way of switching to different cipher.	Yes
16iii	There should be a way to exit the program.	Yes
17	The program should have some way of being able to copy the output into another window.	Yes

Success Criteria Review

Addressing met success criteria

Almost all the success criteria were met in full, no further complications to be addressed until feedback is given from the stakeholders.

Addressing unmet and partially met success criteria

Success Criteria number **6** was not met at all and couldn't be tested for as it was not part of the Design phase, despite being in the success criteria. In the original planning it was only meant to be used if the frequency analysis didn't return the correct answer, but through repeated testing, I found that the frequency analysis program had a high level of accuracy so there was no need to program this feature as it is not necessary to meet the stakeholders requirements as the frequency analysis program already satisfies the stakeholders needs.

If given more time, I could program a brute force program, but it would become redundant due to the frequency analysis technique being more accurate. For a long piece of text, producing 25 possible solutions and running an English checker on all possible solutions would prove very inefficient and time consuming and one of the stakeholder's more general needs was for the program to run within adequate timings, which the frequency analysis program meets.

Therefore, this lack of success criteria being met is due to the success criteria in this area being wrongly defined as it should be 'Checking that the program can successfully crack a Caesar Cipher Encrypted text.'

Success Criteria numbers **8** and **9** were only partially met as it the program can encode and decode a single word, but not anything more than this. So this would need to be rectified in future versions.

Given more time I would correct the coding for the Baconian cipher so it will ignore spaces or any special characters and continue encoding the text, while retaining the spaces between words and retaining the special characters. As well as when decoding the text, it needs to ignore any special characters and continue decoding while keeping the spaces so that when the output can be several different words.

Success Criteria number **17** was met theoretically, but realistically only partially as from the feedback I realize that the reset functionality is not very clear to the user of the program as technically you click the desired cipher button to reset or choose a different cipher, but nowhere in the program denotes this to the user whereas the exit button is clear to the user as it clearly labels an X.

If I had been given more time then it would have been more appropriate to make a separate reset button for each of the ciphers, very clearly showing how the information can be erased from all entry boxes.

Usability Features

The use of buttons meant that none of the stakeholders struggled to navigate the program, except for resetting the current cipher, which isn't too difficult to work out, but it isn't clear either. Overall this was a helpful feature and successfully completed its purpose.

The copy button meant that the stakeholders could copy the output easily and this increased the usability successfully as it made it easier to transfer the data out of the program.

The light yellow background, where normally overlooked, actually increased the usability as it increased the readability of the outputs and made the program easier on the eyes.

Limitations

Algorithms

(1) The Baconian cipher needed more attention during the Implementation stage so that the problem picked up by the stakeholder could be recognised and action could have been taken earlier, but since the basis of the code is already there, it shouldn't take too many amendments to fix the problem.
(2) The Frequency Analysis and Kasiski Analysis could have been put into the Caesar and Vigenère class respectively, but due to the way the solution was developed they are separated.

I also feel like the GUI outlook could have been much more clearly presentable given more time to understand GUI formatting.

(3) The entry boxes where long pieces of text are often entered do not have a text wrap ability so for long pieces of text you can only see the end of the text and it will take more time that necessary to get to the start of the text.
(4) The reset button was not clear, so this could possibly be amended.
(5) Once the text goes over a certain length, not even the text-wrapping can show all the output.

Countering Limitations

(1) In the code for the Baconian class, it needs to be adjusted so that the algorithm for encryption and decryption using the Baconian cipher can accommodate longer piece of text. However, because the Baconian cipher is in a class, it makes this part of the program easier to rectify as the changes will only affect a single class.
(2) This is not detrimental to the workings of the algorithms so doesn't necessarily need to be changed, but it would increase the modularity of the solution.
(3) In the GUI mainframe, a text-wrap ability can be added to all the entry boxes.
(4) In the GUI mainframe, make a separate reset button that clears all the entry buttons and the output sections of the program.
(5) In the GUI mainframe, a scroll feature could be added so that if the output exceeds the length of the window, the user is able to scroll down to reveal the rest of the output.

Maintenance

The Caesar Cipher and Vigenère Cipher will not require maintenance as the workings of these ciphers will not change over time and the running of these algorithms is sound.

The Baconian Cipher requires attention as it has very limited entry features. This needs to be changed in future maintenance updates.

The other limitations can also be addressed in future maintenance. As well as an extra instruction set could be added to help navigation through the program.

It is possible to integrate newly programmed ciphers easily due to the object-oriented modular nature of the program.

Most of the code is also annotated with comments as well as within this document, several walkthroughs and explanations of singular modules can be used to understand the program.

Appendix

Exerts used for Evaluation Testing

Shantaram - Novel by Gregory David Roberts[14]

TEST DATA 1

“In my case, it's a long story, and a crowded one. I was a revolutionary who lost his ideals in heroin, a philosopher who lost his integrity in crime, and a poet who lost his soul in a maximum-security prison. When I escaped from that prison, over the front wall, between two gun-towers, I became my country's most wanted man. Luck ran with me and flew with me across the world to India, where I joined the Bombay mafia. I worked as a gunrunner, a smuggler, and a counterfeiter. I was chained on three continents, beaten, stabbed, and starved. I went to war. I ran into the enemy guns. And I survived, while other men around me died. They were better men than I am, most of them: better men whose lives were crunched up in mistakes, and thrown away by the wrong second of someone else's hate, or love, or indifference. And I buried them, too many of those men, and grieved their stories and their lives into my own.

But my story doesn't begin with them, or with the mafia: it goes back to that first day in Bombay.”

1013 Characters

[14] Shantaram – Gregory David Roberts

Used in Caesar Cipher, Vigenère Cipher

TEST DATA 2

Test Data 1 Caesar encrypted with a shift of 9

“rw vh ljbn, rc'b j uxwp bcxah, jwm j laxfmnm xwn. r fjb j anexudcrxwjah fqx uxbc qrb rmnjub rw qnaxrw, j yqruxbxyqna fqx uxbc qrb rwcnparch rw larvn, jwm j yxnc fqx uxbc qrb bxdu rw j vjgrvdv-bnldarch yarbxw. fqnw r nbljynm oaxv cqjc yarbxw, xena cqn oaxwc fjuu, kncfnnw cfx pdw-cxfnab, r knljvn vh lxdwcah'b vxbc fjwcnm vjw. udlt ajw frcq vn jwm ounf frcq vn jlaxbb cqn fxaum cx rwmrj, fqnan r sxrwnm cqn kxvkjh vjorj. r fxatnm jb j pdwadwwna, j bvdppuna, jwm j lxdwcnaonrcna. r fjb lqjrwnm xw cqann lxwcrwnwcb, knjcnw, bcjkknm, jwm bcjaenm. r fnwc cx fja. r ajw rwcx cqn nwnvh pdwb. jwm r bdaerenm, fqrun xcqna vnw jaxdwm vn mrnm. cqnh fnan knccna vnw cqjw r jv, vxbc xo cqnv: knccna vnw fqxbn urenb fnan ladwlqnm dy rw vrbcjtnb, jwm cqaxfw jfjh kh cqn faxwp bnlxwm xo bxvnxwn nubn'b qjcn, xa uxen, xa rwmroonanwln. jwm r kdarnm cqnv, cxx vjwh xo cqxbn vnw, jwm parnenm cqnra bcxarnb jwm cqnra urenb rwcx vh xfw.

kdc vh bcxah mxnbw'c knprw frcq cqnv, xa frcq cqn vjorj: rc pxnb kjlt cx cqjc orabc mjh rw kxvkjh.”

1013 Characters

Used in Caesar Cipher

TEST DATA 3

Test Data 1 Vigenère encrypted by the letters “sha”

“au mq jakl, il'z a dvny ztgyy, sud s jrgddwk ofl. i ohs s yenvlmaiguajf wzv lgzt zps akesss au hwyoau, a hoidvsgwhwy wzv lgzt zps autwnraay au cjpmw, hnv h pglt ooo dvsl oik zoms if h mseiebm-klcmyilf pjpsgu. wzln a lsuhpwk fjvm loal wrazof, vvwy tzl fjvnl dads, bwawwln ldo ybn-lvwwys, a ieuhmw ty uvufarq'z mgzt ohnlld ehn. dbcc yaf dilo mw hnv mlwd waah el auyokz tzl wgylv ao audah, wzlrw p jgpnwk tzl bgtbsf msmis. p wgykwk ak h gmurmunwy, a ktuynlwy, afk a uvufaejmeaaej. p wsz czhifld gu tzyew jofaiflnlz, bwhtwu, slhbtld, sud kaajcev. p wwut lv wsy. i jhn autg ahw lnwty ybnk. hnv p smyvacev, dhase gahwy mwu ajvufk mw kiwk. tzly olrw ielaej tef ahsu i st, mgzt gm tzlm: tltllr eln oookl lacek dejl cjbnuoev bp au maztsrek, hnv ahjvwf hwsf bq ahw drgug klcgud gm sgtegue wssw'z hsae, gy lgce, gy ifkixmejlnul. afk i tbrald loee, aog taff ox ahgze eln, sud yyiwcev ahwpr kaojpek hnv ahwpr dpvwz ifao ef oou.

bma mq ztgyy vveku't tlgau waah loee, vr optz ahw taxpa: aa ggls thcc ao loal mijzt vhy au bgtbsf.”

1013 Characters

Used in Vigenere Cipher

ERRONEOUS TEST DATA

“Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec gravida sit amet eros sit amet tincidunt. Praesent venenatis ipsum ut volutpat porta. Sed nec arcu ex. Maecenas imperdiet tristique nisl, at dignissim turpis sagittis sed. Pellentesque sit amet nulla eget magna fermentum mattis. Nunc ornare facilisis nibh, id lacinia purus gravida tristique. Nullam eget eleifend magna, at aliquet velit. Praesent diam orci, venenatis eget aliquam vitae, convallis ac mi. Phasellus accumsan posuere eleifend. Nulla congue sem at elit lacinia, vitae tempus purus aliquet.

Curabitur gravida varius aliquam. Maecenas rhoncus, nulla eget aliquam scelerisque, elit arcu commodo enim, dignissim maximus ex massa vitae magna. Etiam eget placerat lorem. Vivamus ac finibus arcu, eget laoreet urna. Nunc mattis vehicula turpis et pharetra. Curabitur id lorem eget eros placerat cursus quis sed lectus. Aenean eget nunc blandit, porttitor libero et, suscipit sem. Donec faucibus lacus vitae luctus vulputate. Aenean dapibus massa ut gravida.”

Approx 1000 characters, 150 words

Used in Caesar Cipher, Vigenere Cipher

Working Done by Hand

In order to understand how to decompose some of the ciphers, some diagrams and processes were produced by hand to better understand how to solve the problem.

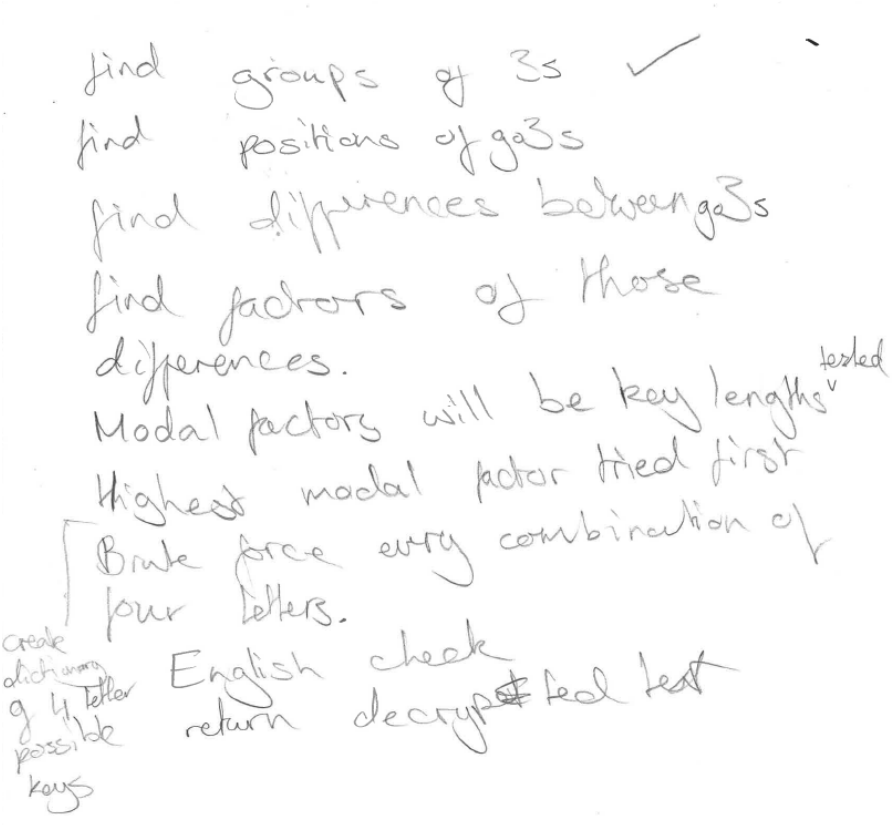


Figure 32 Workings of the Kasiski Analyis

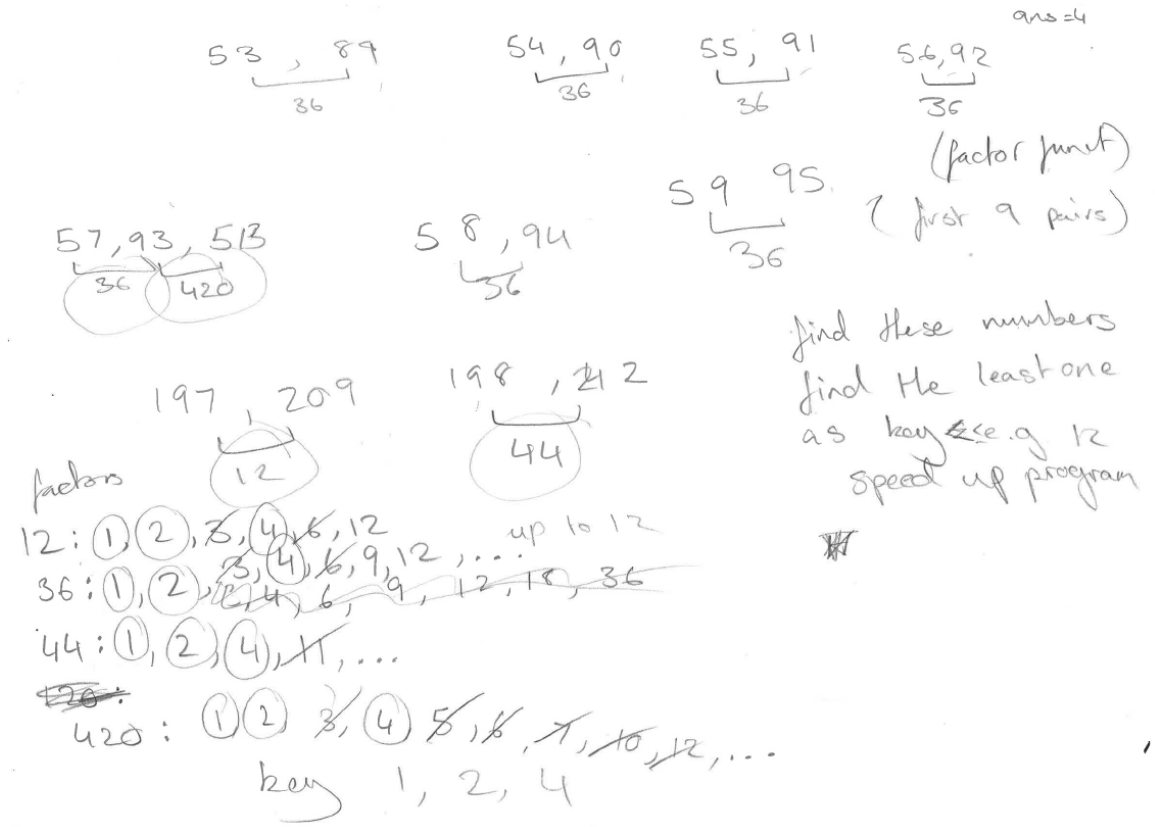


Figure 33 Hand Calculations for the Kasiski Analysis

Sources

1) https://www.cipherchallenge.org/information/rules/
2) https://cryptii.com/
3) https://www.dcode.fr/en
4) https://www.secretcodebreaker.com/history2.html
5) https://blogs.ucl.ac.uk/infosec/2017/03/03/cryptography-basics/
6) http://practicalcryptography.com/ciphers/caesar-cipher/
7) https://learncryptography.com/attack-vectors/frequency-analysis
8) http://rumkin.com/tools/cipher/baconian.php
9) https://www.geeksforgeeks.org/baconian-cipher/
10) https://docs.python.org/2/using/unix.html
11) https://www.cs.cmu.edu/~jbigham/pubs/pdfs/2017/colors.pdf
    Luz Rello, Jeffrey P. Bigham, Human-Computer Interaction, Carnegie Mellon University
    Good Background Colors for Readers: A Study of People with and without Dyslexia
12) Pick-Up – Novel by Charles Willeford
13) http://www.gwicks.net/dictionaries.htm
14) Shantaram - Novel by Gregory David Roberts

Final Code

GUI.py

```
from tkinter import *
from tkinter import messagebox
import pyperclip as cbt      ##clip board tools
import datetime as record   ##imported timer to record times

import CaesarV5 as CC
import VigenereV2 as VC
import BaconianVer0 as BC
import frequencyAnalysisFinal as FA      ##This is Caesar Cipher Cracking
import kasiskiAnalysisFinal as KA       ##This is Vigenere Cipher Cracking

class GUI:
    def __init__(self, root):
        self.root = root
        ##This frame is constant throughout the program
        self.main_frame = Frame(root,bg = "Light Yellow")
        ##These have been initialized as frames and contain information in the
 corresponding functions
        self.caesar_frame = Frame(root,bg="Light Yellow")
        self.baconian_frame = Frame(root,bg="Light Yellow")
        self.vigenere_frame = Frame(root,bg="Light Yellow")
        #These are packed so that the program looks more orderly and these can
 be forgotten and reinstated easily
        self.main_frame.pack()
        self.caesar_frame.pack()
        self.baconian_frame.pack()
        self.vigenere_frame.pack()
        ##These are creating labels to be used later on in the program to disp
lay the result
        self.caesar_result = Label(self.caesar_frame, background = "Light Yell
ow", wraplength=400)
        self.vigenere_result = Label(self.vigenere_frame, bg= "Light Yellow",
wraplength=400)
        self.baconian_result = Label(self.baconian_frame, bg= "Light Yellow",
wraplength=400)
        ##These use grid because within all the frames grid is used, but the f
rames themselves are packed.
        self.caesar_result.grid(row=14)
        self.vigenere_result.grid(row=14)
        self.baconian_result.grid(row=14)

    def CCEncryption(self,caesar_frame,entry1,entry2):
        start = record.datetime.now()
        key = entry1.get()
        if key.isdigit() == False:          ##Validation for key: integer
```

```
            messagebox.showerror("Invalid Key","Please enter the key as an int
eger.")
            entry1.delete(0,"end")
        else:
            key=int(key)
        text = entry2.get()
        caesarE = CC.Caesar(text,key)
        encrypting=caesarE.CaesarCipherEncryption()
        self.caesar_result.config(text=encrypting)  ##This code changes what i
s contained in the label in self with what has been encrypted
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("Helvetica",
 9),text="Copy",command=lambda: cbt.copy(encrypting)) ##This button copies the
 output to the users clipboard
        copytoclipboard.grid(row=15)

    def CCDecryption(self,caesar_frame,entry1,entry2):
        start = record.datetime.now()
        key = entry1.get()
        text = entry2.get()
        if key.isdigit() == False:          ##Validation for key: integer
            messagebox.showerror("Invalid Key","Please enter the key as an int
eger.")
            entry1.delete(0,"end")
        else:
            key=int(key)
        caesarD = CC.Caesar(text,key)
        decrypting=caesarD.CaesarCipherDecryption()
        self.caesar_result.config(text=decrypting)
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("Helvetica",
 9),text="Copy",command=lambda: cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def CCCrack(self,caesar_frame,entry2):
        start = record.datetime.now()
        text = entry2.get()
        caesarC = FA.frequencyAnalysis(text)
        cracking = caesarC.runFA()
        self.caesar_result.config(text = cracking)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.caesar_frame,bg="white",font=("Helvetica",
 9),text="Copy",command=lambda: cbt.copy(cracking))
        copytoclipboard.grid(row=15)

    def VCEncryption(self,entry1,entry2):    ##Validation for key: string, no
numbers, special characters or spaces
```

```
        start = record.datetime.now()
        key = entry1.get()
        if key.isalpha() == False:
            messagebox.showerror("Invalid Key","Please enter the key as a stri
ng of only letters.")
            entry1.delete(0,"end")
        text = entry2.get()
        vigenereE = VC.Vigenere(key,text)
        encrypting=vigenereE.VigenereCipherEncryption()
        self.vigenere_result.config(text=encrypting)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(encrypting))
        copytoclipboard.grid(row=15)

    def VCDecryption(self,entry1,entry2):    ##Validation for key: string, no
numbers, special characters or spaces
        start = record.datetime.now()
        key = entry1.get()
        if key.isalpha() == False:
            messagebox.showerror("Invalid Key","Please enter the key as a stri
ng of only letters.")
            entry1.delete(0,"end")
        text = entry2.get()
        vigenereD = VC.Vigenere(key,text)
        decrypting=vigenereD.VigenereCipherDecryption()
        self.vigenere_result.config(text=decrypting)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def VCCrack(self,entry2):
        start = record.datetime.now()
        text = entry2.get()
        vigenereC = KA.VigenereCrack(text)
        cracking = vigenereC.VCCrack()
        self.vigenere_result.config(text = cracking)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.vigenere_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(cracking))
        copytoclipboard.grid(row=15)

    def BCEncryption(self,entry):
        start = record.datetime.now()
```

```
        text = entry.get()
        baconianE = BC.Baconian(text)
        encrypting=baconianE.BaconianCipherEncryption()
        self.baconian_result.config(text=encrypting)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(encrypting))
        copytoclipboard.grid(row=15)

    def BCDecryption(self, entry):
        start = record.datetime.now()
        text = entry.get()
        baconianD = BC.Baconian(text)
        decrypting=baconianD.BaconianCipherDecryption()
        self.baconian_result.config(text=decrypting)
        end = record.datetime.now()
        print("Timer: ",end-start)
        copytoclipboard=Button(self.baconian_frame,bg="white",font=("Helvetica
", 9),text="Copy",command=lambda: cbt.copy(decrypting))
        copytoclipboard.grid(row=15)

    def CaesarSelect(self):
        self.caesar_frame.pack()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack_forget()
        self.caesar_result.config(text="")
        caesar_frame=self.caesar_frame

        CCnote=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 10),te
xt="Caesar Cipher Input").grid(row=7, column=0) ##This will appear on the wind
ow
        CCkey=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 9),text
="Enter key:").grid(row=8)
        entry1=Entry(caesar_frame)
        CCtext=Label(caesar_frame, bg="Light Yellow",font=("Helvetica", 9),tex
t="Enter text to be processed:").grid(row=9)
        entry2 = Entry(caesar_frame)
        entry1.grid(row = 8,column=1)
        entry2.grid(row = 9, column = 1)

        encryptButton = Button(caesar_frame, bg="white",font=("Helvetica", 9),
 text = "Encrypt",command=lambda: self.CCEncryption(caesar_frame,entry1,entry2
))
        decryptButton = Button(caesar_frame, bg="white",font=("Helvetica", 9),
 text = "Decrypt",command=lambda: self.CCDecryption(caesar_frame,entry1,entry2
))
```

```
        crackButton = Button(caesar_frame, bg="white",font=("Helvetica", 9), t
ext = "Crack", command = lambda: self.CCCrack(caesar_frame,entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid(row = 11, column = 2)

    def BaconianSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack()
        self.vigenere_frame.pack_forget()
        self.baconian_result.config(text="")  ##reseting the result value
        baconian_frame=self.baconian_frame
        BCnote=Label(self.baconian_frame, bg="Light Yellow",font=("Helvetica",
 10), text="Baconian Cipher Input").grid(row=7, column=0)
        entry=Entry(self.baconian_frame)
        BCtext=Label(self.baconian_frame, bg="Light Yellow",font=("Helvetica",
 9),text="Enter text to be processed:").grid(row=8)

        encryptButton = Button(self.baconian_frame, bg="white",font=("Helvetic
a", 9), text = "Encrypt",command=lambda: self.BCEncryption(entry))
        decryptButton = Button(self.baconian_frame, bg="white",font=("Helvetic
a", 9), text = "Decrypt",command=lambda: self.BCDecryption(entry))
        encryptButton.grid(row=11)
        decryptButton.grid (row = 11, column = 1)
        entry.grid(row=8,column=1)

    def VigenereSelect(self):
        self.caesar_frame.pack_forget()
        self.baconian_frame.pack_forget()
        self.vigenere_frame.pack()
        self.vigenere_result.config(text="")
        vigenere_frame=self.vigenere_frame

        VCnote=Label(vigenere_frame,bg="Light Yellow",font=("Helvetica", 10),
text="Vigenère Cipher Input").grid(row=7, column=0)
        VCkey=Label(vigenere_frame, bg="Light Yellow",font=("Helvetica", 9),te
xt="Enter key:").grid(row=8)
        entry1=Entry(vigenere_frame)

        VCtext=Label(vigenere_frame,bg="Light Yellow",font=("Helvetica", 9),te
xt="Enter text to be processed:").grid(row=9)
        entry2 = Entry(vigenere_frame)
        entry1.grid(row=8,column=1)
        entry2.grid(row = 9, column = 1)

        encryptButton = Button(vigenere_frame, bg="white",font=("Helvetica", 1
0), text = "Encrypt",command=lambda: self.VCEncryption(entry1,entry2))
```

```
        decryptButton = Button(vigenere_frame, bg="white",font=("Helvetica", 1
0), text = "Decrypt",command=lambda: self.VCDecryption(entry1,entry2))
        crackButton = Button(vigenere_frame, bg="white",font=("Helvetica", 9),
 text = "Crack", command = lambda: self.VCCrack(entry2))
        encryptButton.grid (row = 11, column = 0)
        decryptButton.grid (row = 11, column = 1)
        crackButton.grid (row = 11, column = 2)

    def main(self):
        welcome=Label(self.main_frame, bg="Light Yellow", font=("Helvetica", 1
1), text = "Welcome \nPlease select a Cipher").grid(row=0, column=0)
        firstbutton = Button(self.main_frame, bg="white",font=("Helvetica", 9)
,text="Caesar Cipher", command=lambda: self.CaesarSelect()).grid(row=2,column=
0)
        secondbutton = Button(self.main_frame, bg="white",font=("Helvetica", 9
),text="Baconian Cipher", command=lambda: self.BaconianSelect()).grid(row= 3,
column = 0)
        thirdbutton = Button(self.main_frame, bg="white",font=("Helvetica", 9)
, text = "Vigenère Cipher", command=lambda: self.VigenereSelect()).grid(row= 4
, column = 0)
        root.mainloop()

root=Tk()
root.geometry("800x600")
root.configure(background="Light Yellow")
root.title("Cybites Cipher")
gui = GUI(root)
gui.main()
```

CaesarV5.py

```
##This version is altered so it works with the GUI

class Caesar:
    def __init__(self, ciphertext, shift):
        self.ciphertext=list([m.lower() for m in ciphertext])  ##ciphertext is
 a list
        self.shift=shift

    def CCProcess(self, rightorleft):
        processlist=[]
        for a in self.ciphertext:
            if a.isalpha()==True:
                b=self.convert(a,self.shift, rightorleft)
                processlist.append(b)
            else:
                processlist.append(a)
        joined="".join(map(str,processlist))
```

```
        return joined

    def convert(self, letter, shift, rightorleft):
        l=chr(((ord(letter)-96)+(rightorleft)*shift)%26+96)
        if l =="`":
            l="z"
        return l

    def CaesarCipherDecryption(self):
        mode = -1
        result = self.CCProcess(mode)
        return result

    def CaesarCipherEncryption(self):
        mode = +1
        result = self.CCProcess(mode)
        return result

'''
#example:
caesar = Caesar("spam",5)
print(caesar.CaesarCipherDecryption())
'''
```

VigenereV2.py

```
class Vigenere:
    def __init__(self,key,text):
        self.fixedKey = [ord(o.lower())-96 for o in key]
        self.key = []       ##this will be key used for decryption or encrypti
on
        self.text = list([m.lower() for m in text])
        keyLengthwr=0   ##with repeats
        for i in self.text:
            if i.isalpha():
                keyLengthwr+=1
            else:
                pass
        for j in range(0,keyLengthwr):
            self.key.append(ord(key[j % len(key)])-96)


    ##This is for Vigenere
    def VCChange(self,rightorleft):
        letterlist = self.text
        numberlist = self.key
        encryptlist=[]
        n=0
```

```
        for i in range(0,len(letterlist)):  ##changes made here: goes through
letterlist and n only increments if the letter is alphabetical
            character = letterlist[i]
            if character.isalpha():   ##validates that letter is in the alphab
et
                keynum=numberlist[n]
                shiftedletter=self.convert(character,(keynum-
1), rightorleft)  ##second part is just the shift
                encryptlist.append(shiftedletter)
                n+=1
            else:
                encryptlist.append(character)
        joined="".join(map(str,encryptlist))
        ##print ("The following is the encrypted text:\n"+joined) #Outputs int
o terminal
        return joined

    def convert(self,letter, shift, rightorleft):   ##same as used in the Caes
ar Cipher
        l=chr(((ord(letter)-(ord('a')-1))+(rightorleft)*shift)%26+96)
        if l =="`":
            l="z"
        return l

    def VigenereCipherEncryption(self):
        mode = +1
        ##print("Vigenere Cipher > Encryption")
        return self.VCChange(mode)


    def VigenereCipherDecryption(self):
        mode = -1
        ##print("Vigenere Cipher > Decryption")
        return self.VCChange(mode)

'''
#example:
test=Vigenere("abc","abc de!!")
print(test.VigenereCipherEncryption())
'''
```

BaconianVer0.py

```
##Baconian Cipher
##using only i and u (if statement will be needed for j and v)
import random
import string

class Baconian:
```

```
    def __init__(self, text):
        self.text = list(text)
        self.ABvalues = {"a":"aaaaa","b":"aaaab","c":"aaaba","d":"aaabb","e":"
aabaa","f":"aabab",
            "g":"aabba","h":"aabbb","i":"abaaa","k":"abaab","l":"ababa","m":"a
babb",
            "n":"abbaa","o":"abbab","p":"abbba","q":"abbbb","r":"baaaa","s":"b
aaab",
            "t":"baaba","u":"baabb","w":"babaa","x":"babab","y":"babba","z":"b
abbb"}
        self.ABkeys = {}
        for key, value in (self.ABvalues).items():
            self.ABkeys[value] = key

##Encryption Algorithms

    def makingAB(self):
        ABstring = []
        letter = self.text
        for character in range(len(letter)):
            if letter[character].isalpha():
                ##print(letter[i])          ##03/02/2020
                ##print(type(letter[i]))
                if str(letter[character]) == "v":
                    changeto="u"
                    ABstring += self.ABvalues[changeto]
                elif str(letter[character]) == "j":
                    changeto="i"
                    ABstring += self.ABvalues[changeto]
                else:
                    ABstring += self.ABvalues[letter[character]]
            else:
                ABstring += character
        return ABstring
        #print(ABvalues[letter])

    #a is lowercase, b is capital

    def randtext(self,ABnfive):
        randomtext=str("")
        for aOrB in ABnfive:
            if aOrB =="a":
                loweralpha=string.ascii_lowercase
                randomletter = random.choice(loweralpha)
                randomtext += randomletter
            else:
                upperalpha=string.ascii_uppercase
                randomletter = random.choice(upperalpha)
```

```
                randomtext += randomletter
        return randomtext

    def BaconianCipherEncryption(self):
        encyptedText = self.makingAB()
        return self.randtext(encyptedText)

##Decryption Algorithms

    def analyseFiveLetters(self):
        upperAndLower = []
        for char in (self.text):
            if char.isalpha():
                if char.islower():
                    upperAndLower.append("a")
                elif char.isupper():
                    upperAndLower.append("b")
        return upperAndLower

    def listOfAB(self,upperAndLower):
        listOfFive = []
        for n in range (int((len(upperAndLower))/5)):
            fiveLetters = []
            fiveLetters.append(upperAndLower[5*n])
            fiveLetters.append(upperAndLower[5*n+1])
            fiveLetters.append(upperAndLower[5*n+2])
            fiveLetters.append(upperAndLower[5*n+3])
            fiveLetters.append(upperAndLower[5*n+4])
            listOfFive.append("".join(fiveLetters))
        return listOfFive

    def comparison(self,listOfFive):
        listOfDecryption=[]
        for ABcode in listOfFive:       ##goes through ab list from input
            for ABalpha in (self.ABkeys).keys():    ##ABalpha is a key in thhe
 ABKEys
                if ABcode==ABalpha:
                    listOfDecryption.append(self.ABkeys[ABalpha])
        decrypted="".join(listOfDecryption)
        return decrypted

    def BaconianCipherDecryption(self):
        baconianObject = Baconian(self.text)  ##returns with B
        a=baconianObject.analyseFiveLetters()
        b=baconianObject.listOfAB(a)
        return baconianObject.comparison(b)

'''
```

```
#example:
def TestDecrypt():
    bacTest = Baconian("aaaBB")
    print(bacTest.BaconianCipherDecryption())
TestDecrypt()
'''
```

frequencyAnalysisFinal.py

```
##Cracking Caesar Cipher: Frequency Analysis

import CaesarV5 as CC
import EnglishCheckerFinal as EC

class frequencyAnalysis:
    def __init__(self,ciphertext):
        self.ciphertext = ciphertext.lower()
        self.tableOfCipherText = {"a":0,"b":0,"c":0,"d":0,"e":0,"f":0,
                    "g":0,"h":0,"i":0,"j":0,"k":0,"l":0,"m":0,
                    "n":0,"o":0,"p":0,"q":0,"r":0,"s":0,
                    "t":0,"u":0,"v":0,"w":0,"x":0,"y":0,"z":0}
        self.sixMostFrequentLetters=["e","t","a","o","i","n"]
        self.dictionary = open("english2.txt", "r").read().splitlines()

    def modalLetter(self):  ##finding most frequently occuring letter
        for letter in self.ciphertext:
                if letter.isalpha()==True:
                    self.tableOfCipherText[letter]=self.tableOfCipherText[lett
er]+1
        mostfrequent=max(zip(self.tableOfCipherText.values(),self.tableOfCiphe
rText.keys()))
        MFL = mostfrequent[1] ##gets rid of unwanted 2D array for 1 element
        return MFL

    def attemptingtocrack(self,modal):
        letterindex=0
        correctDecryption = "not found"
        while letterindex<6:
            attemptKey=ord(modal)-
ord(self.sixMostFrequentLetters[letterindex])
            attempt = CC.Caesar(self.ciphertext, attemptKey)
            attempting = attempt.CaesarCipherDecryption()   ##decrypted text r
egardless of whether it is correct
            wordlist= EC.EnglishChecker(attempting)
            attemptEEng = wordlist.englishcheckerBF()   ##True or False
            if attemptEEng==True:
                correctDecryption=attempting
                letterindex+=1      ##It will continue running through the lis
t so the while loop is satisfied
```

```
            else:
                letterindex+=1
        return(correctDecryption)

    def runFA(self):
        MFL = self.modalLetter()
        cracked=self.attemptingtocrack(MFL)
        return cracked

'''
#example:
ciphertext="Pa tbza ohcl illu hyvbuk h xbhyaly av lslclu. H zhpsvy jhtl pu huk
 vyklylk h jopsl kvn huk jvmmll. P zspjlk h ibu, qlyrlk h myhur vba vm aol ivp
spun dhaly, ulzalk pa, wvbylk h ohsm-
kpwwly vm jopsl vcly aol myhur huk zwypurslk pa spilyhssf dpao jovwwlk vupvuz.
 P zjypiislk h joljr huk wba pa if opz wshal. P dvbsku'a ohcl yljvttluklk aol
buwhshahisl tlzz av h zahycpun hupths. Aol zhpsvy dhz aol vusf jbzavtly, huk h
maly ol hal opz kvn ol slma. Aoha dhz aol lehja tvtlua zol lualylk. H zthss dv
thu, ohyksf tvyl aohu mpcl mlla. Zol ohk aol mpnbyl vm h alluhnl npys. Oly zbp
a dhz h isbl adllk, zthyasf jba, huk vcly oly aopu zovbsklyz zol dvyl h mby qh
jrla, ivslyv slunao. Apuf nvsk jpyjbshy lhyypunz jsbun av oly zthss wplyjlk lh
yz. Oly ohukz huk mlla dlyl zthss, huk dolu zol zlhalk olyzlsm ha aol jvbualy
P uvapjlk zol dhzu'a dlhypun huf ypunz."
test = frequencyAnalysis(ciphertext)
print(test.runFA())
'''
```

kasiskiAnalysisFinal.py

```
##Cracking Vigenere Cipher: Kasiski Analysis
##Will not accept keys that are 1

import EnglishCheckerFinal as EC
import VigenereV2 as VC

import string
import itertools

class VigenereCrack:
    def __init__(self, ciphertext):
        self.ciphertext=ciphertext
        textNS=""
        self.spaceindex=[]
        for single in range(0,len(self.ciphertext)):
            character=self.ciphertext[single]
            if character.isalpha():   ##will return true or false
                textNS+=character       ##adds any character that is an alphab
etical character to a new string
            elif character.isspace():   ##will return true or false
```

```
                self.spaceindex.append(single)    ##adds the index of all the
spaces to a list
        self.textNS=textNS.lower()      ##makes every letter in the text lower
case with no special characters
        #To be used in multiples or whatever it is called now
        self.possibleKeyLength=[]

        self.threelist = []

        self.listOfFactors=[]
        ##used in factorsList
        self.differences = []
        self.occurdict={}
        ##used in likely keys
        self.likelyKeyNum=[]
        #used in VCCFreqA
        self.keyProbably=True       ##True is just a placeholder
        self.possibleKeyList=True
        self.decryptedText = "not found"        ##Set to this value by default

    def getMultipleIndexes(self,searchElement):
        indexesList = []
        indexPosition = 0
        while True:
            try:
                indexPosition=self.textNS.index(searchElement, indexPosition)
                indexesList.append(indexPosition)
                indexPosition+=1
            except ValueError as e:
                break
        return indexesList

    def repeatedThrees(self):  ##looks at three characters and compares them t
o every other three and records the index of repeated keys
        incrementfromzero=0
        incrementfromthree=3
        threedict = {}

        for number in range(0,(len(self.textNS)-2)):
            three=self.textNS[incrementfromzero:incrementfromthree]
            if three in threedict.keys():  ##if three is already in the dictio
nary
                self.threelist.append(three)  ##appends three if it is repeate
d
                threedict[three]+=1     ##this tells how many occurances there
 are
            else:
```

```
                threedict[three]=1  ##adds it to the dictionary if its not the
re
            incrementfromzero+=1       ##runs after the if statement
            incrementfromthree+=1
        return self.threelist

    ##This returns all the differences between every group of 3
    def differenceBetweenGroups(self):
        for stringOfThree in self.threelist:
            templistOfPos=self.getMultipleIndexes(stringOfThree)
            ##print(templistOfPos) ##e.g. output: [53,89]
            for posNumber in range(0,(len(templistOfPos)-1)):
                ##print(templistOfPos[posNumber+1]-
(templistOfPos[posNumber])) ##e.g. output: 36, 420
                self.differences.append(templistOfPos[posNumber+1]-
(templistOfPos[posNumber]))
        return self.differences

    #This is finding factors that could possibly be the key
    def factor(self,x,min_factor):
        ##x is fac from the for loop in factorsList
        for i in range(1, x + 1):
            if x % i == 0:
               if i<=min_factor:
                  self.listOfFactors.append(i)
        return self.listOfFactors

    def factorsList(self):
        factors_factor=[]
        ##print("Min Value: ",min(self.differences)) ##This value is the minim
um number produced from the differences between groups of threes
        for fac in self.differences:
            templist=self.factor(fac,min(self.differences))
            for num in templist:
                factors_factor.append(num)
        ##every occurance of factors up to the min factor

        for fac in factors_factor:
            if fac not in self.occurdict.keys():
                self.occurdict[fac]=factors_factor.count(fac)
        if 1 in self.occurdict.keys():
            self.occurdict[1]=0
            ##removes 1 character shifts as they are caesar cipher and commonl
y disrupt the most common factor

    def likelyKeys(self):       ##creating a list of likely keys
        likelyKeyList=[]
        highest = max(self.occurdict.values())  ##upperbound for values
```

```
        lower = highest-1000                    ##lowerbound for values
        for num in self.occurdict:
            if (self.occurdict[num] <=highest) and (self.occurdict[num]>lower)
:
                ##print(num,self.occurdict[num])        ##prints the ones that
 satisfy the range
                likelyKeyList.append(num)
        likelyKeyList=sorted(likelyKeyList)     ##sorts them in numerical orde
r
        likelyKeyList=likelyKeyList[::-
1]       ##highest key returned is the most likely key but will have several i
n this list
        self.likelyKeyNum=likelyKeyList[0]      ##Will return higheset one
        return self.likelyKeyNum

    def VCCFreqA(self):                #Vigenere Cipher Cracking Frequency Ana
lysis
        sixMostFrequentLetters=["e","t","a","o","i","n"]
        listOfStrings=[]
        mightBeKey=[]
        ##creating lists of letters for every letter of the key length
        for magnitude in range(0,self.likelyKeyNum):
            mightBeKey.append([])
            listOfStrings.append([])
        listTextNS=list(self.textNS)
        for letterpos in range(0,len(self.textNS)):
            listNum=letterpos%self.likelyKeyNum
            listOfStrings[listNum].append(listTextNS[letterpos])
        listOfDict=[]
        for lists in listOfStrings:
            dictOfAlpha = {"a":0,"b":0,"c":0,"d":0,"e":0,"f":0,
                    "g":0,"h":0,"i":0,"j":0,"k":0,"l":0,"m":0,
                    "n":0,"o":0,"p":0,"q":0,"r":0,"s":0,
                    "t":0,"u":0,"v":0,"w":0,"x":0,"y":0,"z":0}
            for letter in lists:
                dictOfAlpha[letter]+=1
                mostCommonLetter=max(dictOfAlpha, key=dictOfAlpha.get)
            for popularLetter in sixMostFrequentLetters:
                tempkey=chr((ord(mostCommonLetter)-ord(popularLetter))+97)
                if tempkey in string.ascii_lowercase:
                    mightBeKey[listOfStrings.index(lists)].append(tempkey)
         ##This makes a 2d array e.g. possiblekey=[[first letter of key],[seco
nd letter of key],...]
        self.keyProbably=mightBeKey
        return self.keyProbably

    def keyCompile(self):
        def finished(variables, arr):       ##arr = array
```

```
            return variables[0] == len(arr[0])
        def updateVariables(variables, arr):
            carry1 = True
            currIndex = len(variables) - 1
            while carry1:
                variables[currIndex] = variables[currIndex] + 1
                if variables[currIndex] == len(arr[currIndex]) and currIndex !
= 0:
                    variables[currIndex] = 0
                    currIndex = currIndex - 1
                else:
                    carry1 = False
            pass
        ##      [[possible 1st letters of key],[possible 2nd letters of key],.
..]
        ##arr= [["a", "d", "e"], ["h", "i"], ["q", "w", "e"]]   ##=self.keyPro
bably
        possibleKeys = []
        variables = []
        for i in self.keyProbably:
            variables.append(0)
        possibleKeys = []
        while not finished(variables, self.keyProbably):
            key = ""
            for i in range(len(self.keyProbably)):
                key += self.keyProbably[i][variables[i]]
            updateVariables(variables, self.keyProbably)
            possibleKeys.append(key)
        self.possibleKeyList=possibleKeys

    def keyCracking(self):  ##Trying possible keys
        decryptedText=""
        n=0
        for key in self.possibleKeyList:
            n+=1
            vigeneretest=VC.Vigenere(key,self.ciphertext)
            VDtest = vigeneretest.VigenereCipherDecryption()
            test = EC.EnglishChecker(VDtest)
            check=test.englishcheckerBF()
            if check:
                self.decryptedText=VDtest
                break
        return self.decryptedText

    ##This module should work but there is something wrong with my Vigenere de
cryption program

    def VCCrack(self):
```

```
        ##print(self.textNS)
        self.repeatedThrees()
        self.differenceBetweenGroups()
        self.factorsList()
        self.likelyKeys()
        self.VCCFreqA()
        self.keyCompile()
        answer=self.keyCracking()
        return answer

'''
#example:
##key = abcd
text2 = "Io ob cbuh, iu'u d lppj suquy, bpg a dtrwegg oog. L wbu d rfxrlvvlooc
uy xjr lpuw hju ldfcos jp kesqln, b rkimqvoqjhr xjr lpuw hju lnugjrjvb io euin
g, dne c sofv zhp nrsu jls tqxl jp d mbzlmvo-
vedwuiua srjurn. Xjhn J gvcbrhd gtrm ujdt qtlspp, rvft whf huoov zamn, eeuyheo
 vzo hwq-
tpyhrt, K eedcpe na fovpwrz'u potv zaovhd ncq. Lven rbp ziuj pe bpg fmgz wjvk
mf cfrpuv tig zosng tp Kqdjc, zhfth I kqlnff whf Drmccb mbhla. J yrrlgg at c j
uotxnogu, a toxghnhr, bpg a dqxnuguffkwes. K zat ekajphd pp whsgh cppwiogqtt,
dhaugq, sucebff, dne uwasxhd. J yhnu vr wbt. L rbp lnuq whf gqena juou. Dne K
vusxlvff, zhjnh oujhr ngq asqxne oh djgg. Tigb wfth bfvwes ohn ujdn J cp, mpuw
 og vken: dhtugu mfp zhpuh ljxhs xgue dtxndjhd vr ln nkvtbmhs, bpg titrwo czaz
 db tig zrppj sferne qi spohoog hltg'v hbvh, os nrvf, qu ioflfggueoeh. Aof L b
vtlee vken, vro ncqy ph whpuh mfp, dne iuifxhd ujhis uwoskhs bpg tiglr mkyet k
qtp ob oxp."
test=VigenereCrack(text2)
print(test.VCCrack())
'''
```

EnglishCheckerFinal.py

```
##English Checker Program
##english2.txt is a dictionary of words, this must be located in the same dire
ctory/ folder as this program.

class EnglishChecker:
    def __init__(self,text):
        self.text=text
        a = open("english2.txt", "r")
        with open("english2.txt", "r") as dict:
            a = dict.read().splitlines()
        self.dictionary=a

    ##makes text lowercase with all spaces retained
    def processTextLCaWS(self):
        text=self.text.lower()
        processedText = ""
```

```
        for character in text:
            if character.isalpha() == True:
                processedText+=character
            elif character.isspace() == True:
                processedText+=character
        return processedText

    ##BF = Brute Force
    def englishcheckerBF(self):
        wordlist=self.processTextLCaWS().split()
        point=0
        totalwords = len(wordlist)
        for word in wordlist:
            for line in self.dictionary:
                if word == line:
                    point+=1
                    break
        if (point/totalwords)>0.90:     #Over 90% of text is English
            #Program runs fast providing it doesn't print the outcomes
            return True
        else:
            return False

'''
#example:
text1 = "her last smile to me wasn't a sunset. it was an eclipse, the last ecl
ipse, noon dying away to darkness where there would be no dawn."
pieceOfText=EnglishChecker(text1)
print(pieceOfText.englishcheckerBF())
'''
```

english2.txt

Can be downloaded from:
http://www.gwicks.net/dictionaries.htm

English:

ENGLISH - 84,000 words	**QTYP** 143Kb	**TEXT** 223Kb
ENGLISH - 194,000 words	**QTYP** 332Kb	**TEXT** 512Kb
UK ENGLISH - 65,000 words	**QTYP** 117Kb	**TEXT** 178Kb
UK ENGLISH - 82,000 words	**QTYP** 140Kb	**TEXT** 219Kb
USA ENGLISH - 61,000 words	**QTYP** 108Kb	**TEXT** 166Kb
USA ENGLISH - 77,000 words	**QTYP** 131Kb	**TEXT** 206Kb