ADA

Cryptography Assistant

Lucy Jefford

A Level Computer Science NEA

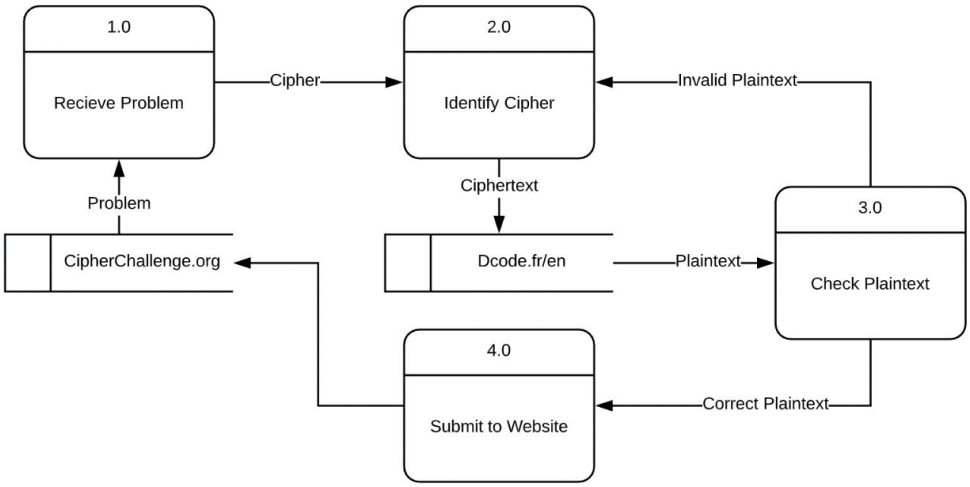
Table of Contents

Analysis

Introduction

At St Mary's Calne, we have a wide selection of clubs and supra-curricular activities, one being a Code Breaking Club. This happens in the Autumn Term and follows the Cipher Challenge[1], entering the codes once we've cracked them. This is a nationwide competition set by the University of Southampton, where each Thursday there is a new encrypted text released to the website. The quicker you crack it, the more points you get. As a group, there are a range of levels of experience, from advanced to beginners who've only just learnt what a Caesar Cipher is. As the competition develops, the codes increase in difficulty, often involving a new type of cipher or by adding something devious like reversing the text.

Data Flow Diagram



This how we are currently solving the ciphers that we receive. As you can see, we often use another site or application, with Dcode.fr being the example here. However, I believe this can be improved.

End User

Dr Drape, a Computer Science and Maths teacher, runs our club. He has his own program to quickly solve (or brute force if necessary) a large majority of the codes from the Cipher Challenge: Caesar Cipher, Substitution, Vigenère, Rail Fence, Transposition, to name a few. He would like a program for this club to be able to use, to aid their problem solving and make it easier to solve some of the more regular codes that crop up in the challenge. He's got a background in Cryptography too, however his specialty (for coding) is based around Functional Programming, an extremely different way of thinking and writing programs. It would be useful for him to have the project in a different language, especially the teaching language of this school – Python.

As well as running the club, he teaches Computer Science at A Level. On our current course there are some aspects of encryption involved in the course, and which means this sort of project is relevant to his teachings. At the end of this, my project could be useful to show methods of solving certain ciphers, however it is more relevant to the Code Breaking Club, as they experience a greater number

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

of ciphers, and more often. In the syllabus, only Vernam and Caesar cipher are mentioned: the former of these is impossible to break with modern computers, the latter the easiest.

Interviews

In my first meeting with Dr Drape (SD), I discussed potential solutions, and requirements for the project.

Here is a small extract:

What is the most important thing in the project for you?

SD: The ability to solve ciphers, without a human doing too much trial and error would be best.

What would you prioritise, better cipher solving or a faster, more optimised program?

SD: In any program, you want it to be optimised, however if there is optimisation to the detriment of greater cipher capabilities, this would not be ideal.

How would you best recommend identifying optimal decoding algorithms?

SD: For certain algorithms there are distinct and well-known solutions, for example Caesar is the most well-known. Frequency analysis is often extremely useful and should be the first thing you should do. It'll let you know if it's a direct English (transposition), a monoalphabetic (Caesar/substitution) so one letter is mapped to one letter in the alphabet, and a "flat distribution", so likely a polyalphabetic (Vigenère).

Which ciphers do you most want to solve?

SD: Transpositions, substitutions but semi-automated.

Do you care about a GUI or just a command line interface?

SD: I don't care, but for the club an easy to use interface would help them, but the actual solver is most important. Focus on that.

Survey

To gather information from my entire user population, I decided to create a survey to best create a program to suit their needs. I used Survey Monkey[2] to design one, which I have put in the Appendix for full reading.

The data I got from this was varied and helped guide my design proposal for this project, especially towards the ways of approaching the ciphers.

Data Analysis

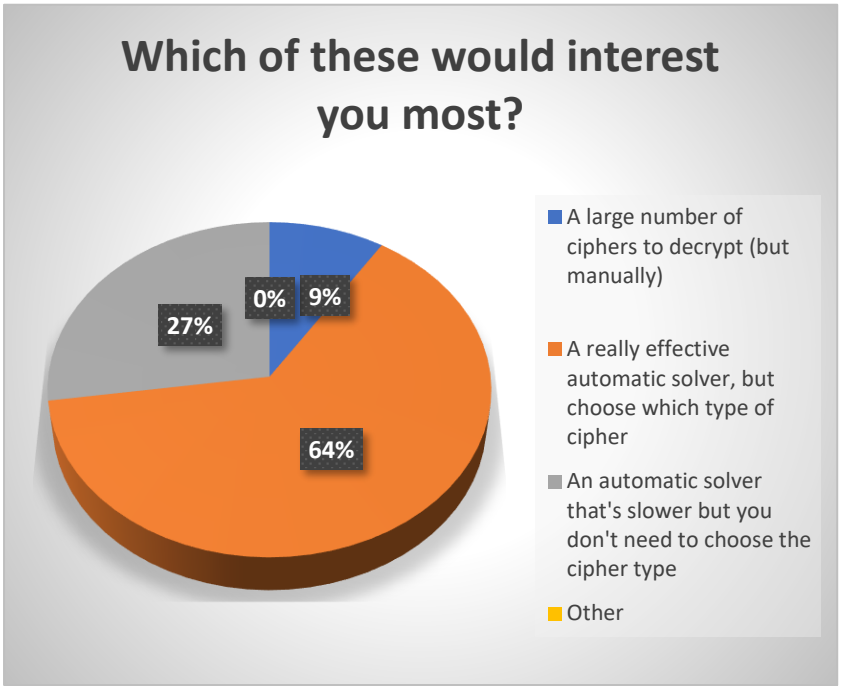


Figure 1

In Figure 4, you can see that a large percentage (91%) of people want a program that can solve a cipher automatically, but 64% want to have to specify the cipher, in order to have a faster program. I will prioritise individual automatic decoders, rather than a general program to solve all, from these results.

9%, who are in the minority, would like to decrypt ciphers manually, so I will still include options to decode these ciphers through user trial and error, to cater for all opinions.

In Figure 5, the data shows that most recipients would like to learn how to break the ciphers by themselves, so a guided tutorial would be preferable. This is interesting to me, as I would not have prioritised this aspect; I assumed they would want to just receive the answer, so I will incorporate this into my potential system further.

This will change my design and involve me working with a select few simple ciphers and creating an interactive interface to teach the user how to manually crack the code.

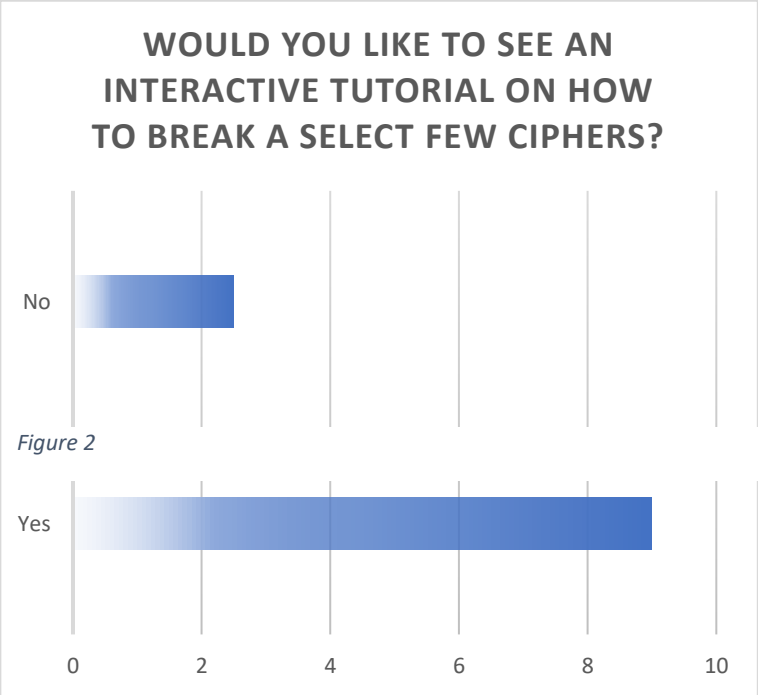


Figure 2

[2] https://www.surveymonkey.com/

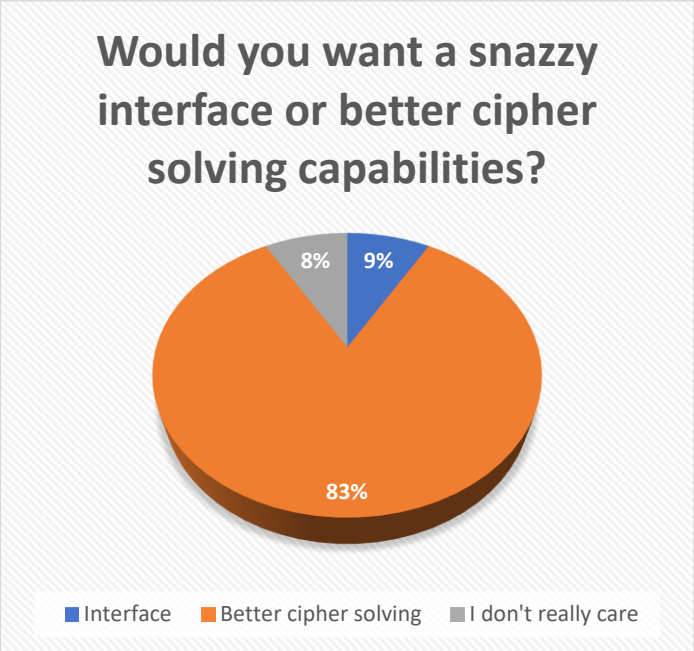


Figure 6

In Figure 6, it shows the answers to "Would you want a snazzy interface or better cipher solving capabilities?". From this pie chart, we can see many of the responses want a better cipher solver.

This agrees with my thoughts already, as my end user would prefer a system that has better cipher solving. So, my design concept hasn't really changed much from this, only confirming my hypothesis.

From this survey, it's been really interesting to see the different thoughts and ideas of the Club. It has really sped up the interviewing process – and anonymised it too. I hope to be able to put some auto-solving capabilities into the code, but if I cannot, then I'll work that in another way, as the focus here seems to be on the cipher-solving capabilities. I'm also not considering using a GUI anymore, as the users don't seem to require a "snazzy interface" from my data.

Investigation Methods/ Research

For my project, I will research each individual cipher focusing on how to encode and decode it, along with the relevancy to this project. I will also look at the methods the club members already use, from trial and error to specialised websites that can automatically solve ciphers extremely quickly.

Caesar Cipher

Often a code breaker's first cipher, the Caesar Cipher is the most basic of codes. It involves shifting every letter in the code by a certain number of places in the alphabet. It's the weakest code as there are only 26 permutations that a code breaker must run through, therefore computation-wise it's extremely easy to break.

As you can see in Figure 3[3], the alphabet wraps around in a circle, and by knowing the shift, or one corresponding letter, the entire code is now able to be deciphered. This automatically makes it extremely easy, along with the set amount of permutations.

Figure 3

However, this cipher is found quite often in the Cipher Challenge, especially towards the start of the challenge. This is because the whole point of the project is to encourage kids to code break, and at the start they make sure they use ciphers that are easy to recognise and decode. There are multiple practice rounds, with ciphers that are there to just show you what it's like, and you don't get any points for solving or not solving them.

Caesar ciphers with a shift of 13 (aka ROT13) are involuntary ciphers, which means you do the exact same thing to them to get back to the starting plaintext.

[3] https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/CipherDisk2000.jpg/1920px-CipherDisk2000.jpg

Substitution

A substitution can be a monoalphabetic cipher. This involves pairing each letter with a unique letter in another alphabet. It can be filled with characters other than the natural alphabet, however examples like the table below are the most common.

Actual	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M

This table above is an example of a substitution alphabet. So, if you had “COMPUTING IS FUN”, this would translate to “EGDHXZOFU OL YXF”. To reverse the process, you find the letter in “Cipher” and write instead the “Actual” letter.

This is a non-cyclical cipher, even if the algorithm and variables may be the same. You could find an involuntary cipher in a substitution, if the alphabet was symmetrical, i.e. if B = W and W=B. Then you could apply the same algorithm twice without changing the variables or method.

This type of cipher is quite easy to spot as it has an obvious frequency analysis. If you look at Figure 2, the ciphertext in this case appears to be a monoalphabetic, and E has most likely moved to G, and T or A has moved to J. This is because the normal frequency of E appears to be over G. This is how most substitutions are solved, plus a bit of intuition about the exact pairings if the same percentages of letters are involved.

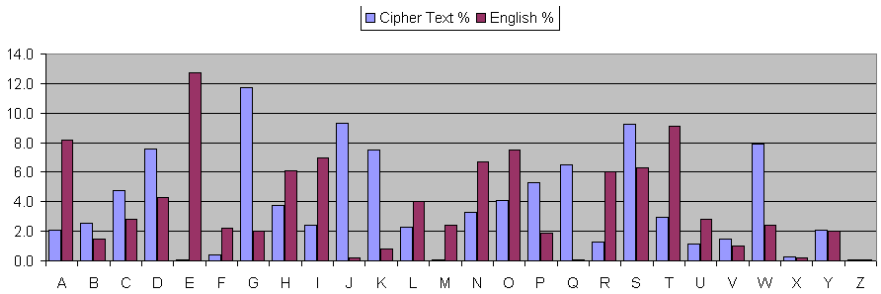


Figure 4

This is one of the best ways to identify the correct pairings, by picking the next most frequent letter on the ciphertext frequencies. However, this is rife with problems because in some cases letters won’t be present at all – for example in the book “A Void” where there is no “E” s in the whole text.

The best solution would end up suggesting possible replacements to the characters, rather than replacing and trying to solve it automatically, as this cipher requires lots of lateral thinking, drawing upon sources of knowledge that would be hard to automate.

Vigenère

A Vigenère cipher is based upon the idea of enumerating each letter and being able to add that value onto an existing one (looping around if you go past Z). In practice, A = 1 and Z = 26, and quite often the way you would get these numbers in programming is by using the inbuilt ASCII values to convert letters to numbers (and vice versa) each time.

To decode a Vigenère, you need the key, or a section of the text known as a "crib". A key can be any length, but usually between 3-6 letters long. The longer the potential key length the more crib you need.
Decoding using the key is easy, you simply minus the current key value off the letter value.
Having to use a crib is a little more difficult. You simply use the crib as the key, displaced at various points, and if any repeating words crop up then it's likely that is the key. However, this works best with a long crib, and if the crib is too short then we will have a much worse guess at the key. The reason this works is because you add the values, so if you minus the actual text it'll leave the key. You can see an example of decoding a Vigenere in Figure 5.

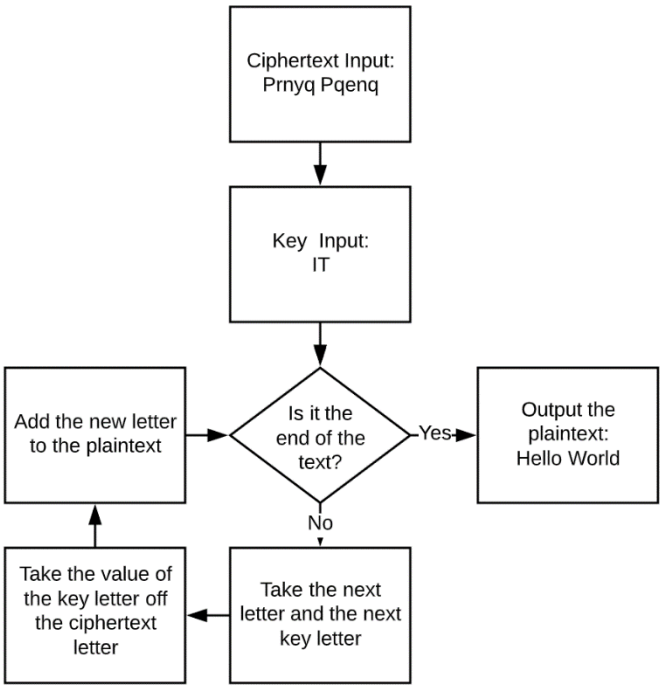


Figure 5

To suggest multiple key lengths (or to see if you are close to an English-like text) you can look at the Index of Coincidence (IoC). This is a number that shows the predictability of the text, or how likely you are to pick the same letter again. English has a usual value close to 1.73 however it varies per text and by what words you use: for example, there is a book called the Void which uses no e's in the entire text, which would throw off a normal human trying to solve it but would not majorly affect the IoC as there is many other letters being used; and too small of a text would turn out an IoC far off 1.73.

You can calculate the IOC using the formula: $IoC = \frac{\sum_{i=1}^{c} n_i(n_i-1)}{N(N-1)\times c}$. However, this seems very daunting to see it in the algebraic format. In order to calculate the Index of Coincidence, you find the frequencies of each letter in the text, and for each letter, multiply it by the frequency -1. You add up all the multiplied values, then you go on to calculate the length of the text multiplied by the length -1, which you subsequently multiply by the randomness constant, c, which is 0.0385.

Rail Fence

In Figure 6, we have a table showing how a Rail Fence cipher is set up. As you can see from this photo, it is also known as the Zig Zag cipher, due to its unique shape. In this example, dots represent spaces, which are included in the formation of the cipher.

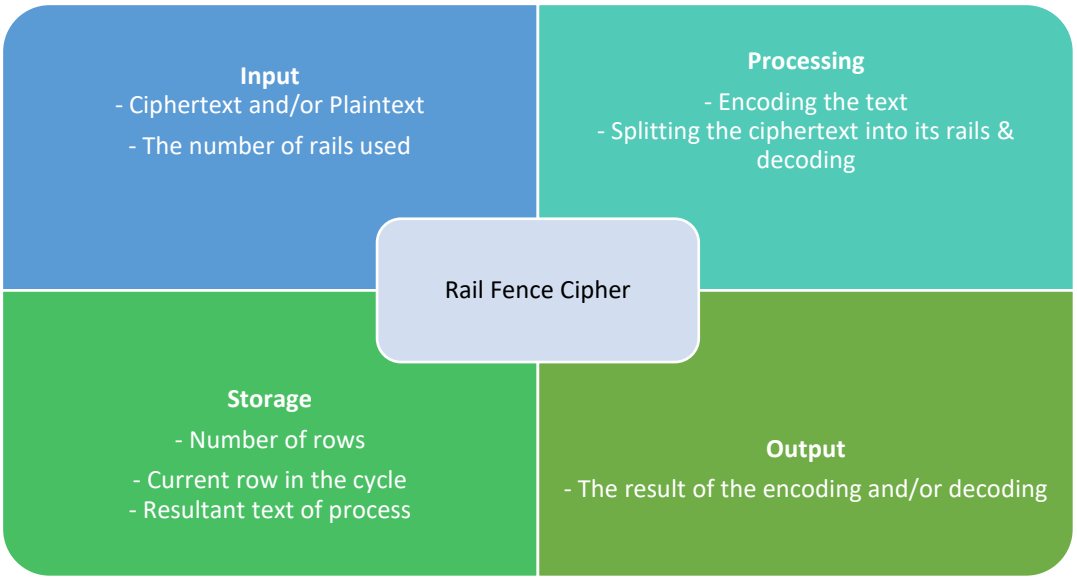
C						I						.			
	O				T		N				S		F		
		M		U				G		I				U	
			P						.						N

Figure 6

This text of “COMPUTING IS FUN” goes to “CI OTNSFMUGIUP N”, which happens when you read along the rows, and by putting the text in the name-sake zigzag pattern.

Due to this cipher’s method of manipulating the order of the text, rather than the letters, we get a normal frequency analysis, making this a transposition style cipher. Except this is unique because the distance to each next letter is subjective. So, to re-order this example, we have 1-4-9-14-10-5-2-6-11-15-12-7-3-8-13-16. In this, we have a mix of a 3,4, 5 gaps, and the gaps can be positive or negative. This means the codebreaker needs to be aware of when they are in the cycle to adjust their next jump accordingly. This makes it harder to decode than a basic transposition or columnar, where there is one set jump number.

Each number of rails has a different amount of characters per cycle. To calculate this you need to use the formula: $Cycle = (Number\ Of\ Rails\ \times 2) - 2$ [4] You minus 2 because the top and bottom rails only add one letter to the cycle, rather than the usual 2.



This diagram above describes the information required in a Rail Fence Cipher, along with some computational information like what the user needs to remember to decode it.

[4] https://en.wikipedia.org/wiki/Rail_fence_cipher#Solution

Transposition

C	O	M	P
U	T	I	N
G	.	I	S
.	F	U	N

There is a couple of ways of laying out a transposition, but both give similar outcomes. In each, you read perpendicular to which you write, so for example if you read down, you right along. For a 4x4 example, both ways return the same answer: you read "COMPUTING IS FUN" to be "CUG OT FMIIUPNSN". You take every Nth letter, so in this case, read every 4th until the end, then skip back to the start, and start from the second letter, and so on. There is a standard number, and it doesn't change as you go.

C	P	I	.	.	N
O	U	N	I	F	X
M	T	G	S	U	X

However, for the other example, we use 6x3, and because it is bigger than the actual cipher length, we have two extra characters to fill it up. This reads as "CPI••NOUNIFXMTGSUX". This has an Nth value of 6, so every 6th character is picked.

C	O	M	P	U	T
I	N	G		I	S
	F	U	N	x	x

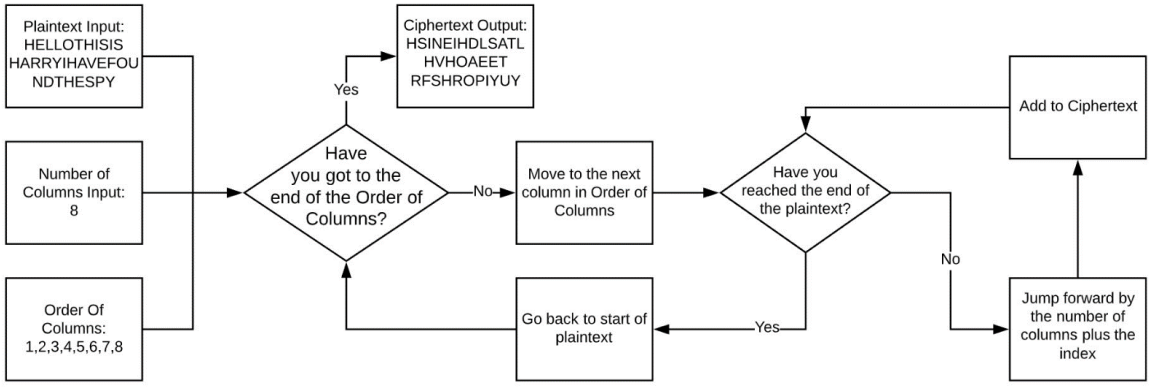
For my project, I will be focusing on a columnar transposition, in which you write along but read down, as this is harder to both encode and decode. So, in our example from above, it'd be encoded by "CI ONFMGUP NUIxTSx" (with the x's as filler letters).

An extra layer with the transposition is that often there is a key associated with the columns or rows. This means that whilst there is still an Nth value, it is mixed up in a weird order. This key is the columns or rows numbered and then mixed. For example, in this third example, we have jumbled the columns, resulting in a ciphertext of "ONFUIxTSxP NCI MGU", with a key of 1-4-5-3-0-2. To decode this text, we no longer need just a simple Nth number, but the key with that too.

O	U	T	P	C	M
N	I	S		I	G
F	x	x	N		U

If you wanted to try to decode this without the key, you would have to try all combinations then decide which looks closer to English. There is a mathematical formula to deciphering this, where you give each value of the key an integer number, then using each number you rearrange in such a way that when you pick that order, it'll decode the key. So, in the case of the example above, you'd get a "Reverse" key of 4-0-5-3-1-2.

Encoding a Transposition Cipher

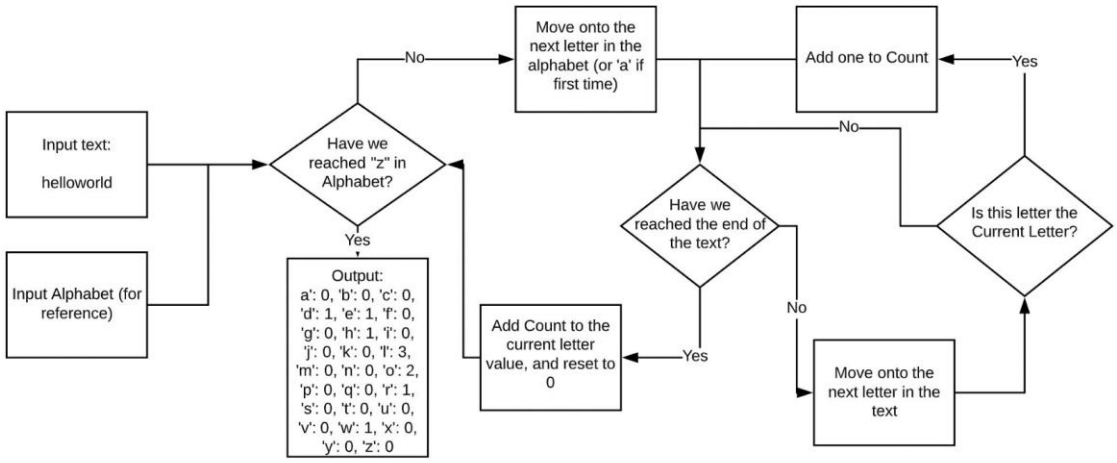


Analysis

Whilst you try and figure out how to decode or solve the cipher you are dealing with; a codebreaker's most important tool is their analysis and analytical capabilities. Most often, the client is looking for simple identifiers like letter frequencies or the length of the text, which can indicate specific key facts about the cipher.

For the frequency analysis, the main aim of this is to simply find all occurrences of that letter and repeat for every letter in the alphabet. Most of the time you find this either with percentages ($letter\ count\ \div text\ length$) or just the flat frequency, so I aim to have both of these in this project.

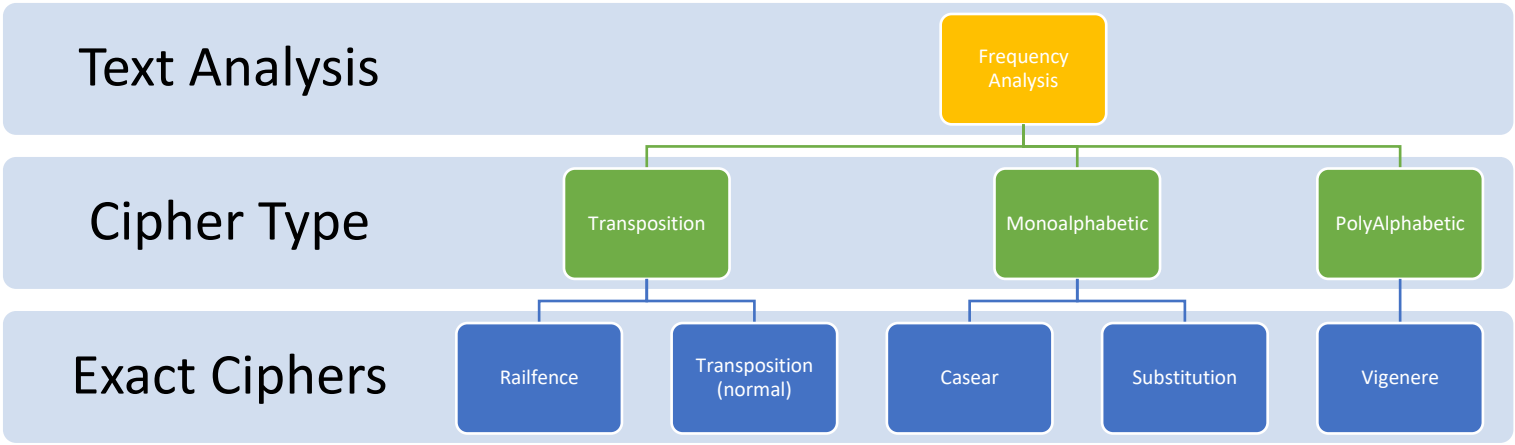
A Letter Frequency Analysis



Something else that is really useful is the creation of bi-grams (or tri-grams etc.). This is when you take the pairings of letters and find the most frequent combinations of the letters. You can do this in two ways – a sliding bigram or a rigid one. For example: "abcdefg" would be "ab", "bc", "cd", "de", "ef", "fg" (for a sliding bigram) or "ab", "cd", "ef" "g" (for a rigid). I'll also aim to put this into the project, as a codebreaker will look for a tool like this when they are decoding unusual ciphers. Mostly, they will look for a pairing of 2,3,4 or 5 letters, but potentially they may want larger pairings, so I'll look into doing a program that can handle any number for the pairings (or X-grams).

Other useful analysis techniques are comparing the standard English Frequency Analysis side by side to the cipher's Frequency Analysis. Normally they use the percentages in this situation as they are easier to compare, and also can be visually seen as larger/smaller. This only works on very large pieces of text, as in a small piece the percentage can be influenced a lot easier. For example, if I did a frequency analysis on "Hello World" (as above), you get percentages that are nothing like English simply because the text is not representative of the English Language. If I were to use an extract of a book however, then the frequency analysis would seem "better" or closer to the normal percentages. This is also why the Index of Coincidence cannot be used with smaller text sizes.

A code breaker also needs to be able to identify the cipher that they are looking at. This diagram helps breakdown the ciphers into subsections to subsequently pinpoint the correct one.



You would begin by taking a frequency analysis of the text. In the frequency analysis is "flat"[5] , then it appears to by a polyalphabetic cipher; if the frequency analysis looks remarkably similar to English, but instead over the wrong letters, it could be a monoalphabetic cipher. And if it exactly fits English (or close enough) it may be a transposition cipher[6]. From this cipher type, you can then identify the cipher using other methods, liking looking at the length of the text or the Index of Coincidence value.

[5] All values are extremely similar
[6] Not necessarily a transposition, but a shifting of the same letters

Other Systems

Throughout the internet, and in many a book, there are endless options for websites to crack your Vigenère or your substitution cipher, the only problem is the considerable number of sites that you must visit.

One other system is the website Dcode[7]. This can decode many ciphers automatically, with a small crib or key, and can try dictionary attacks against ciphers like Vigenère or Beaufort. This website is used by multiple of the club members, I included, when presented with ciphers that we don't know. However, if you don't know the specific cipher it's very hard to use this website, as it has very efficient algorithms but very specific ones.

Figure 7 – a screenshot of Dcode's Caesar cipher decoder

Another system is Dr Drape's own Haskell program. For his thesis, he wrote a paper (and therefore code) to solve multiple ciphers using functional programming. This is extremely high level, and the syntax of the functional program is hard to understand, especially for beginners or people who haven't studied this style of programming. Also, the rules of the challenge say that teachers can help or assist but not give the answers, and by allowing us access to this code, then he's crossing a line in the rules.

[7] https://www.dcode.fr/en

Current System and Problems

Description

During the Autumn Term, when the project is on, the Code Breaking Club meets once a week in one of our school's boarding houses. They each come with a laptop, to access the codes and use online material to solve them. They talk it through as a group, seeing if we can identify the type of the cipher, and using analysis tools online to help guide our thoughts.
One we think we have decided what type (monoalphabetic, direct English, polyalphabetic) we try to identify exactly which cipher it is. This can often be difficult because there are many variations of each cipher, for example the Beaufort[8].

After we think we have our cipher, we either break it by hand or by using a website. This depends upon the cipher. If it's a substitution, the most common approach is by using Word, and referencing a sheet of frequency analysis. For other ciphers like a Vigenère, we would use a cipher solver tool online, that would then take the job of solving it away from us.

If this cipher only returned nonsense, we'd look at other types of this cipher, or look at different ways to manipulate the text, such as reversing it or applying a cipher twice. Then we'd go back to the start and reattempt our solving.

Once we had reached a plaintext that look more like English, we'd share it via an email and the team leader would submit it to the Cipher Challenge website.

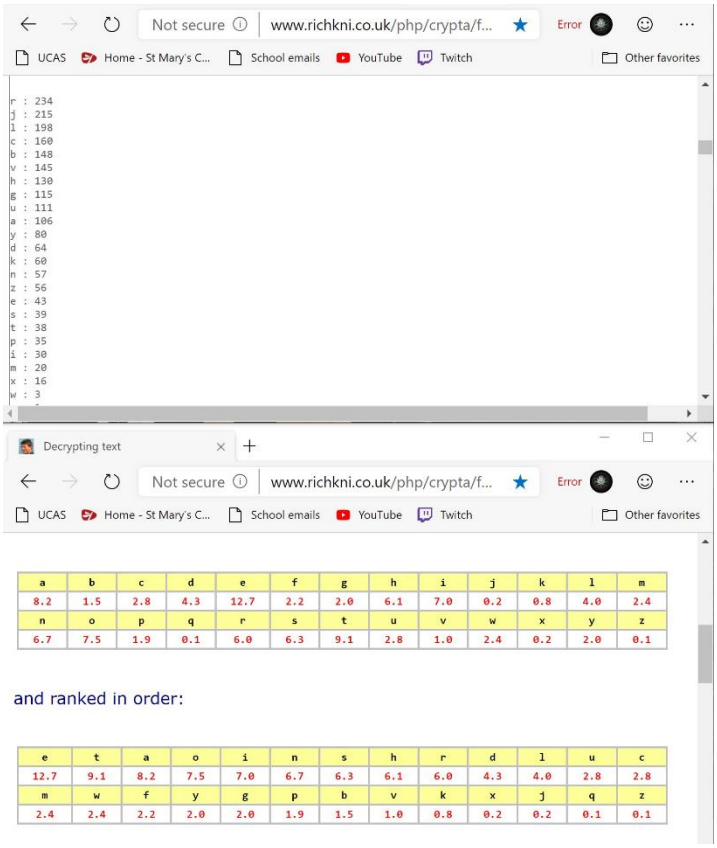
[8] For more information on the Beaufort see

Current Cipher Solving Methods

Here is a sample of the methods of solving the ciphers that the club members already use.

This is an attempt at solving a substitution in Word. The real version is in lower case, whereas the unbroken ciphertext is in capitals. You can see the method, with some obvious words already figured out. This cipher is actually 4A from 2018's challenge[9].

the DYKe NaH in the naZe. the IYaDX ZKHeKZ iH NoGYA SaZoKH aH the in-hoKHe ZKHeKZ oS DGiZe at neN HDotYanA PaGA. the hiHtoGP looXH (oG at YeaHt, NiXiEeAia) toYA Ze that it haA Ieen Het KE IP inHEeDtoG neaZe anA ED GanAaYY HoZe tiZe aSteG aEGiY 1875 aH a NaP to KHe the EGiHoneGH' EGoEeGtP HtoGe to inHtGKDt neN GeDGKitH in the aGt oS AeteDtion. the naZe SiGHt aEEeaGH in EKIYiD in an aGtiDYe in the oIHeGMeG neNHEaEeG in 1877, anA the hiHtoGianH HaP it NaH DoineA aH a TGiHYP heaAYine IP a WoKGnaYiHt Nho NaH HnKIIeA IP neaZe, IKt i aZ not Ho HKGe. it AiAn't HeeZ an aDDiAent that AoKTYaH IYaDX NaH YooXinT to Het KE a toE HeDGet aGDhiMe in YonAon at that tiZe, anA HDotYanA PaGA TaMe hiZ the EeGSeDt YoDation. it NaH DentGaY anA eaHP to aDDeHH. it haA HeMeGaY entGanDeH, Nith a HteaAP SYoN oS MiHitoGH, not aYY oS theZ entiGeYP GeEKtaIYe, that NoKYA AiHTKiHe the DoZinTH anA ToinTH oS the aTentH anA oSSiDeGH oS IYaDX'H netNoGX. IeHt oS aYY it haA the entiGe ZetGoEoYitan EoYiDe SoGDe HtanAinT TKaGA, NhiDh NaH TGeat SoG IYaDX, anA toKTh on Ze. it NoKYA Ie the iAeaY YoDation SoG the HhaAoN aGDhiMe, HDotYanA PaGA haA ZoMeA to neN EGeZiHeH HeMeGaY tiZeH HinDe it NaH SiGHt Het KE, IKt i NaH IettinT that the HhaAoN aGDhiMe haA ZoMeA Nith it, anA i HEent the IeHt EaGt oS thGee NeeXH HDoKtinT the noGZan HhaN IKiYAinTH anA the neN "neN HDotYanA PaGA" neOt AooG at the DKGtiH TGeen IKiYAinT. it iH not eaHP to Ao that NithoKt IeinT HEotteA, henDe the niTht tiZe GeDonnaiHHanDe in the Gain. the NateG anA the DoYA EYaPeA haMoD Nith ZP ZooA, IKt aYHo Nith ZanP AeteDtoG HPHteZH Ho it NaH NoGth the AiHDoZSoGt.
no-one NaH YiXeYP to TiMe Ze the GeaY EYanH oS the EYaDe, thoKTh haGGP NaH haEEP to EGoMiAe Ze Nith a DoEP oS the oSSiDiaY YaPoKt, anA i KHeA a YiAaG HPHteZ to ZaE aH ZKDh oS the oKtHiAe aH i DoKYA. DoZEaGinT the tNo HhoKYA haMe GeMeaYeA KnZaGXeA HtoGaTe aGeaH anA TiMen Ze HoZe iAea hoN i ZiTht Tet in, IKt nothinT HhoNeA KE KntiY i haA a YKDXP IGeaX. YiteGaYYP. i haA YoNeGeA the YiAaG AeteDtoG AoNn a HhaSt on the DKGtiH TGeen GooS, hoEinT to Tet a TYiZEHe into HoZe oS the neaGIP GooZH. it haA WKHt aloKt GeaDheA TGoKnA YeMeY Nhen the nPYon Yine HnaEEeA. it ZaAe a heYY oS a GoN aH it DYatteGeA AoNn, anA i EGeEaGeA to GKn, IKt no aYaGZH Nent oSS anA, ZoGe iZEoGtantYP, i notiDeA HoZethinT HiTniSiDant. it haA taXen aGoKnA SiMe HeDonAH SoG the YiAaG to DGaHh AoNn, NhiDh, at a GoKTh eHtiZate, Zeant it haA SaYYen SGee SoG aGoKnA one hKnAGeA anA tNentP ZetGeH. that NoKYA haMe taXen it a YonT NaP KnAeGTGoKnA, anA theGe NeGe no IaHeZent GooZH ZaGXeA on the EYanH in that YoDation. i NaH EGettP HKGe i haA SoKnA the HhaAoN aGDhiMe: noN i WKHt haA to SinA a NaP in.



Another way is using websites to frequency analyse the cipher given. Here is a screenshot of how they compare the normal English values with the ciphertext:

You can see here they have to have 2 tabs open to see the English values, as the website[10] they use shows either English or your ciphertext's values. From this, I can see that putting these values on the same page/outputting them at the same time would be more efficient and help them to optimise their cipher solving.

[9] https://www.cipherchallenge.org/challenges/35230-2/
[10] http://www.richkni.co.uk/php/crypta/freq.php

Data Volumes

In general, the size of text given by the Cipher Challenge is a medium length piece of text. An example that I have found is Cipher 4B (2019), which was a transposition; it is just under 2,500 characters in length and is a valid example for the average ciphertext used.
This in particular doesn't affect my complexity, but I may want to test with longer or similar sized texts to test the time taken to decode or perform the relevant processes, as that'll give an idea of how long it'll take the code to do the relevant challenges.

The time complexity of my deciphering solutions should be computable. One more letter added to the text should only add one more period of time, as the solution should go through the text changing or placing every letter, so it should increase linearly, which is the best I could hope to achieve.

The exact length of text for some ciphers is relevant. For example, the way the solution would work for a transposition, your text should be a multiple of the columns, otherwise you would get an incorrect solution. A similar case occurs with a Rail Fence solution, where you need a full set of rail cycles, otherwise when you decode the solution, you'll end up with a slightly jumbled solution.

Here is an example of the length of text we get given each time we are released a cipher. It is not a small amount of text, however there is not too much to be unmanageable. Usually there is between 2000-3000 characters each cipher.

4B Ciphertext:

OOHSEERRWCOITUEIEHPIELTINFUTVEOGNRTWDENIIETIGCOATOTIETRCLTLRSESEBHCEOTEOTDONHNETTECNS
NCVTAITTUEEUITLAUOTRYDHTLHNURCRITIOAFCMADVSESATIJSTTHNBDOANTTEEEAETHAHILRGEIEEGYNDFTOF
REOSKNAHSRLELSTDOODAFLIIIAARNHELBYEEHLHLGHSCTHTBOIUNPTELTRLTESMSIGPTDAEECRAEUDCOLONSE
GOFACVALMTBLJNBOHIOAIRNHTMPESRAFABARDGMOEXTESRIGHLSEDERPNGRTEIDEBOIOWRSOILTASRNJRHM
TFOTTGNSLHCDSHITRTESLIHNCIWNOPRYEATLSIETEUOYAEEOTGQFNTLSMNEFNRABLWCMUEYHIREEIDEDUTTN
NTAOSSNLRNLARBAHNHOERNHDTLOEMDLNNQFIHEECIRFAUARMAPAOEREMEEHSSGSPVRYEDETIIRTEOLYESHS
IWCKHNTCOMHSFIOICELEICOADWLOUTCBHEORRATCPELREORPHAFAOETSNFITRSIETYNMIAENTHILAOVTMHH
ULICAPEIRPELHEOTFAFDOLRSHHYHAREQEHDFPIYTHOISTSOJRMTRUNEBHUHIRLENTLOEILSTICEAYTUNNITNNEA
OTNEIGNEISIRLMPAHHSETDCRHNHARTHMWNFRFCSIDIEOIDIEEAVOEROELSEDDHRTTTTSEEHNNSIRIDPNOIIEPU
ORUEHDETTEOIGUTUYTOODIEEODCHREHHSEAOFMLEHWVSRTRHTSENANIPTVSRISHOEDRELTMTDTOUIDEYER
ATSERRLNHEGMAEELSETSAIDTLTETDOIHLFPRGAECEOPLHNTAOIEAEMVNPHOSPTIAPLNGSIERTCOEHOPHLTUDL
FEKDSCEGEIFARTTLTRUTEPTENAIRCAITEAUHEFSEABIDATTTMUUMIIOTVLPPGYCRTOEOCSAOVRRPAIEIALAAQN
EVFNOTRTNRIMFVAGINFRFRMMBNEUUAUTACNDATILMURCRDCOTALGNSAOYRNLHLMETBRDTLRIMFTNTGNT
TUTEOEYOHIBWCAEDHEEECNTEUAYIRTONNCIITECLTEHETPTRPUEHMHLNROEYANSEVRDCALSTTUBEUSCTPESG
ONEHMNCOEIEDLSHWUCLNTAIEEIASITIEUTSGOAFCEESEAEULHVHEFEHHAARONIOFAEORSAOEIIAUNAISTLOTH
UITAOTTNUNOPNRSENEIHONJIPYGTPLBEHOEOEOSSTNHXHTCEETRBAIILATYTIDTNMBEUEYFXBPSFLSEEROEIIEA
SEHYEIOEILADAOFSESETSFRONRNRNTEVUOOEYOISIHMUNEAECSUEOTLEJAEENELEBEGDENDOGSOEKMHRDAT
RTEYHNEDEHNROVQRRWOMROAACLOHACNHPGIMGUEBSTITCITCRLNTIHIPEUMUSASOEKAAIEMLTNCODSAER
NHHNLEURGOAUIRTHGVWBSEARHHOCNLIOSRTOFMHHETETTSLTCEOTCODLIMNEAEHTVYWBSISSSEHHAPLSMI
RHEAFASEOTIRIRTTLEAYHTUDREDTTFSIBNOITATSSEODHEPAXLAPFRCVSNIRAEDHROOSMOYPGRMECHWSSEAH
TRHCCRRFLEPNMEVEENESAWEEEUESSRDTTHRDOISTMDIEHRNLBUVOOHXEDWESODARNNVOEINOWVASTTIOL
OADLTALNHTEUNEETOTPOTSDMRRLTPIETSOWCNDHAANXFIRNSEHUARTTNTRCMNHAWVICTHGCBCIAIUTDVOI
TSAUKAENCEHDUAPAEEOOFMITCRIILTSERSLIATCIHRDOCEGALTSCINDLAORPTDDHTTOEGESOFGICOIIOVFGMET
RISISYIDTARLERRGISEUULCMEETNITLAETUSBTITJMTYOFNSATEEIEAONHSTTELAOTFTNYEHWTOSASOCITTAEAO
LPLEOCHLTRICSPNHGUBTOIEODITTIVRIIOORRPEOOMQNBRTSHUSTORRTEIAECLTIOTATEFNRTAEHDNRMUATT
UBDIRDTFERSTROEIITSHEDOTREIMTMHEOOHUEIDGTHBMAUCTEIOTHEALOAPAEIESMSLADENKFTPOICPIHREAF
COMHLRLIEFFAYNDURSDAYDLDOAHGPERIRNRSGAISANFDOOEULEGICRRNIOLIILFRARRENIAMAESETNGUNAIO
ORTOIRSTSFUEOBHARSRIEDANNDDCHLIST

4B Plaintext:
apolloxipostflightanalysisspacecraftreentrytrajectoryreviewthisdocumentcontainsinformationabouttheeventsduringthereentryphaseofapolloxiithasbeencompiledtosatisfyflightplanningsafetyrequirementsthereviewcoversthefollowingmissionphasetransearthcoastwhichendswithreentryintothemidpacificrecoveryareatheoriginalflightplancalledforseparationofthecmfromthesmtwelveminutesbeforeentryinterfacesuccessfulseparationwouldtheniggerasequenceofburnsintendedtostabilisetheservicemoduleandtomoveitoutofthereentrycorridortoavoiddebriscollisionswiththecommandmoduleduringthereentryphasethereactioncontrolsystemburnsequenceinvolvedboththerollandtheminusxjetstheattitudeburnswereintendedtosetuparollinordertostabiliseremainingfuelandtopreventuncontrolledgyrationduringtheboostthatwouldhavethentakenthesmoutofthereentrycorridorintoahighaltitudeorbitthatwoulddecayonlyafterthecommandmodulehadlandedintheeventthesequenceofburnsdidnotachievetherequiredtrajectoryshiftwithreportsfromtheastronautsonboardandfromacommercialairlinepilotthatthetwospacevehiclesreenteredtheatmospheretogetherwiththecmpassingthesmduringtheplasmaburnphasegiventheproximityofthevehiclesduringreentryitisconsideredhighlyfortunatethatnodebrisfromtherelativelyunprotectedsmstruckthecmandsimulationsshowthatsuchastrikewouldhavebeenlikelytocausecatastrophicdamagetothecmpossibledefectsarisingfromacollisionincludeiheatshielddamageevenaminorcrackinoneoftheheatshieldpanelswouldbelikelytocausesuperheatingwhichcouldbreachthehullleadingtofurthercatastrophicdamagetothevehiclewithprobablelossoflifeiiprematurefiringoftheparachutepyrotechnicsleadingtofullorpartiallossofthedescentarrestsystemandlossoflifeiiidamagetotheparachuteshieldmighthavepreventedthepyrotechnicsfromreleasingtheparachutesleadingtoacatastrophiccollisiononlandingandlossoflifeivdamagetooneormoreoftheparachutescouldhaveledtoahighvelocityimpactwithprobableresultinginjuriesandpossiblelossoflifethehighlikelihoodofcatastrophicfailurearisingfromthedeviationfromflightparametersmeansthatfurtheranalysisoftheseparationburnstrategyisrequiredpendingthatremediationfortheapolloxiireentrytrajectoryisahighpriorityitissuggestedthatsurplusfuelshouldbeejectedfromthesmbeforetheseparationburnbeginsinordertostabilisetheattitudeandthattheminusxburnsshouldbetimedtocoincidewiththerolljetstoimprovestability

Problems

After looking at my current system, and the points highlighted by my interview with Dr Drape, there are a few problems in the current system that do need a solution:

- The members are using multiple websites and having to use up time and substantial amounts of resources, because they do not have everything in one place
- When you are substituting letters individually via Word or other tool, you can potentially incorrectly change a letter to one already in the text, completely running the decoding, and thus waste a lot of time. This would be easily optimised, and the user might find it a slicker/easier operation
- If you get a completely new cipher, where do you start? This project could also assist with not knowing the type of cipher, or where to start on identifying the type of cipher
- Users may not understand how to solve the ciphers, especially beginners, or those who haven't seen the cipher in a long time. A piece of code that walked them through the process could help them understand what they are doing.
- Dr Drape has code in Haskell, but it's not easy to access or understand for an end user who doesn't know the language or syntax, and he cannot provide it due to the rules of the competition.
- When you are analysing a piece of text by hand, you easily make mistakes or incorrectly assume something. These simple mistakes made by human errors, could be easily avoided by using a program, hiding the complexity and calculations from the user.

User Requirements

1) There should be a selection of ways to analyse the text to aid the user to identify the type of the cipher.
    a) You should include some frequency analysis capabilities
    b) You should include the index of Coincidence, and an explanation of what it means (to the user)
2) You must have a tutorial mode available, to walk the user through a few simple ciphers, for example Substitution and Caesar, to guide the less advanced codebreakers through their first ciphers.
    a) If you can have a blind decryption too, that would be useful
3) A semi-automatic solver for a set of ciphers. These include Caesar and Substitution.
    a) The user must input the text
    b) If the text does not have an automatic solver, in the tutorial, there must be some assistance with solving it.
    c) If possible, provide tools to assist with description, even if there is no automatic solver (for example with Rail Fence, Transposition, etc)
4) A way to decrypt manually for each cipher
    a) Caesar, Rail Fence, Transposition, Substitution, Vigenère
5) A way to encrypt for each cipher
    a) Caesar, Rail Fence, Transposition, Substitution, Vigenère
6) It must be easy to use and understand as it is for a variety of user abilities

Acceptable Limitations

1) No time limit from the program, in acceptable boundaries, for example in orders of a few minutes.
2) Lots of possible answers for an automatic solver
    a) The automatic decoders will work heuristically, suggesting possible outcomes, and potential matches, but leave it to the user. You can only have a best guess scenario.
3) I don’t need to consider collaborative working
4) Does not work for other languages, it will only be valid for English
5) If the user enters text/files, it will be in the English alphabet – won’t work for binary.

Objectives

1) Encrypt a selection of Codes
    a) Caesar
        i) Allows for a plaintext input
        ii) Allows for a number to shift by
        iii) Go through all letters, shifting by a set number
        iv) Converts each letter into ascii, performs calculation, and converts back
        v) Shift number must only be accepted between 1 and 26
        vi) Outputs a ciphertext
    b) Substitution
        i) Allows for a plaintext input
        ii) Allows for an alphabet input (any values)
        iii) For each letter in the plaintext, there must be a corresponding letter in the new alphabet.
        iv) Must only accept when a full alphabet is used
        v) No repeated letters/unique one to one relationship
        vi) Goes through plaintext, replacing each letter with allocated value
        vii) Use different cases (lower/upper) to denote plaintext or ciphertext
        viii) Outputs a ciphertext
    c) Vigenère
        i) Allows for a plaintext input
        ii) Allows for a key input (only letters)
        iii) For each letter in the plaintext, it cycles the key infinitely and adds the alphabet value of that relevant key letter to the plaintext letter
        iv) Only changes letters (upper or lower) but this doesn't affect the cipher output. So skips the spaces that have punctuation or spaces in, without affecting the key.
        v) Outputs a ciphertext
    d) Rail Fence
        i) Allows for a plaintext input
        ii) Allows for a Rail number input
        iii) Goes through, taking every letter, adding it to the rung below, then stepping back up the lines (repeating for the number of rails and length of text)
        iv) Outputs a ciphertext
    e) Transposition
        i) Allows for a plaintext input
        ii) Allows for an amount of columns
        iii) Allows for an order of said columns (must contain all possible integer numbers between 0 and number of columns)
        iv) Must split the plaintext into set columns
        v) Must be able to stitch the columns back together into the ciphertext
        vi) Outputs ciphertext.
2) Decrypt a selection of Codes
    a) Caesar
        i) Input ciphertext
        ii) Input the shift number
        iii) Use the Caesar encoding algorithm, but with a negative shift
        iv) Output plaintext
    b) Substitution

i) Input ciphertext
ii) Input cipher's alphabet
iii) Use Substitution algorithm but with right way round of alphabet (see encoding substitution)
iv) Output plaintext

c) Vigenère
i) Input the ciphertext
ii) Input the key
iii) Use Vigenère base algorithm, but subtracting key letter (rather than adding it) and therefore a separate function will be necessary

d) Rail Fence
i) Input the ciphertext
ii) Input the number of rails
iii) Use the Rail Fence algorithm, cycle down

e) Transposition
i) Input ciphertext
ii) Input number of columns
iii) Input key order
(1) In some specified syntax, e.g. 1-2-3-4-5 or 1 2 3 4 5.
(2) Validate this input carefully, no greater than number of columns etc
iv) Take every nth + key and repeat until the end of the message.
(1) E.g. if key was 1-3-5-4-2, letter order would be 1,3,5,4,2,6,8,10,8,7… etc
v) Output plaintext

3) Analyse Text
a) Frequency Analysis
i) Input text
ii) Output frequencies of each letter
(1) In percentages
(2) In actual number (integer) values
iii) Identify and output most common 3, 4 and 5 letter combos
iv) Compare to normal English values
(1) Close to English ranges (Transposition)
(2) Same English numbers, different letters (monoalphabetic)
(3) Not at all similar (polyalphabetic)
(4) Output this comparison

b) English Values
i) Table/Information on percentages of each letter in the UK language, ranked.
ii) Able to be displayed

c) Index of Coincidence
i) Work out using the IOC algorithm - $IoC = \frac{\sum_{i=1}^{c} n_i(n_i-1)}{N(N-1)/c}$
ii) The average IoC is a multiple of key length

4) Auto Solve a specified cipher
a) Caesar
i) Run through all 26 possibilities

b) Substitution
i) From the frequency analysis suggest possible replacements
(1) Suggest character matches, i.e. If J is the highest, suggest it might be an E.

(2) This will not physically solve it for the user, only suggest replacements and potential matches

5) Tutorial Option for less experienced code breakers
   a) Each option will have a brief explanation on how to break before they start/see the ciphertext
   b) Walkthrough Option
      i) This will walk the user through what the computer is doing, explain in words what each algorithm is doing.
         (1) There will be text for each step, along with the changed ciphertext, showing how it develops over the time period
      ii) An inbuilt set available, with Caesar, Vigenère, Rail fence, and an unknown cipher.
      iii) There must be a selection of test texts, of varying lengths. Use extracts from books or poetry or speeches, something that's well known.

Dialogue with Users

Date	Type	Comment
June 2019	Conversation	Discussed potential ways of solving the problem, initial user requirements suggested
June 2019	Interview/Survey	Sent a survey to the club members and client
June 2019	Interview	Narrowed down the User Requirements
October 2019	Conversation	Confirmed the requirements and proposed solution fit their needs (at a meeting of the coding club)

Considerations of potential solutions

For this project, there are many languages I could do this in. Because I aim to write this using Object-Oriented Programming, a more OO based language might be preferable (like C# or Java). And because there is a lot of maths involved, something like a language built for functional programming (i.e. Haskell or Wolfram) might be preferable. However, there are a few of reasons why I will use Python instead.

My end user, Dr Drape, has written something like this code before, but in Haskell. He doesn't want a rewrite of his code, but something which is more accessible and easier to read. The readability of Haskell is extremely poor to a non-programmer, and to aim his project to the club better, he would like it accessible. Python is extremely accessible, especially when a GUI is involved; also, for the programmers that would use this, Python is the language taught at St Mary's Calne in Computer Science therefore making it easier to understand.

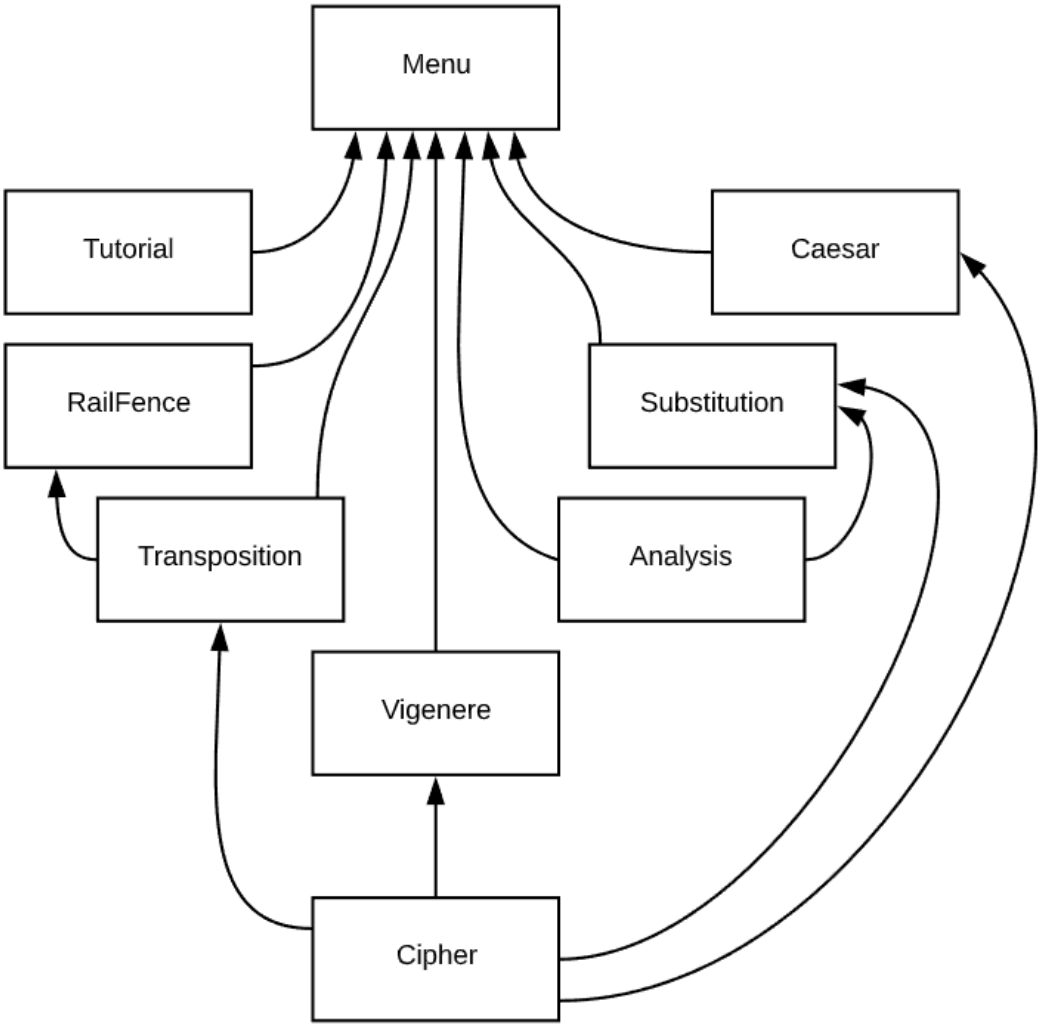
I am most confident in using the syntax of Python, and as it is the taught language at my school, I will have more support and assistance writing the project. I know the libraries and modules well and learning another language at this stage would result in a less than optimal project at the end of it. I do not want to sacrifice complexity of code or solution; due to the fact I don't know the language well enough.

Design

Design Explanations and Ideas

When designing this project, I had the main idea that each of the suggested classes dealt with one cipher each, and the various information that it needed is encapsulated in the class itself. This will continue to work in my favour, as this will help ease of testing later as I can separate individual pieces of code and test them without affecting other code, and later on perhaps extend the solution. I wanted to utilise a natural language style interface and menu as in my research for this project I came along a mixture of tutorials and simple tools but wanted the user to feel this felt more homely or welcoming.

Here is a hierarchy chart on how I would like my code to set out. It involves 9 separate entities, which are connected in several ways.



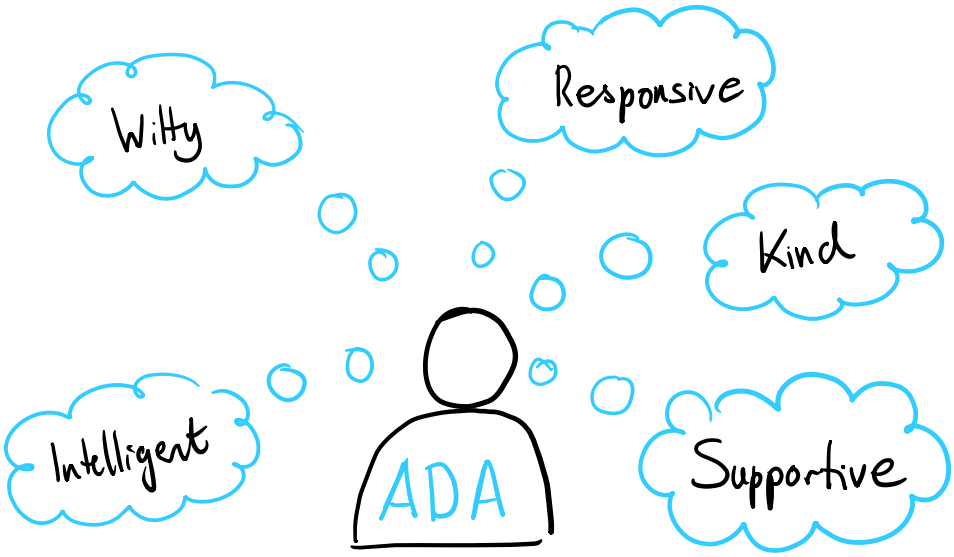
In my design section, I explore each of these sub-sections, as they are going to become my individual classes for the project. In this, I expand on the reasoning behind some of the design choices and why they are split up in such a way. I also run through some of the more complex algorithms and describe how they are going to function in the code, as well as how the user experience will affect my design.

Natural Language

Personality

An interesting idea for this project is the use of natural language ideas to help encourage ease of flow in the menu system and a better user experience for those new to codebreaking. I had an idea of using a personality, that guides the user through their menus and the codebreaking process, in order to provide a semblance of continuity and hopefully encourage more users to feel welcome in a club or hobby that may feel very analytical, cold, and harsh.

So I came up with ADA, which stands for Advanced Decryption Assistant, which will be a very polite and hopefully extremely helpful 'bot' that can guide you through a tutorial, analyse your cipher well, and encode or decode it at will.



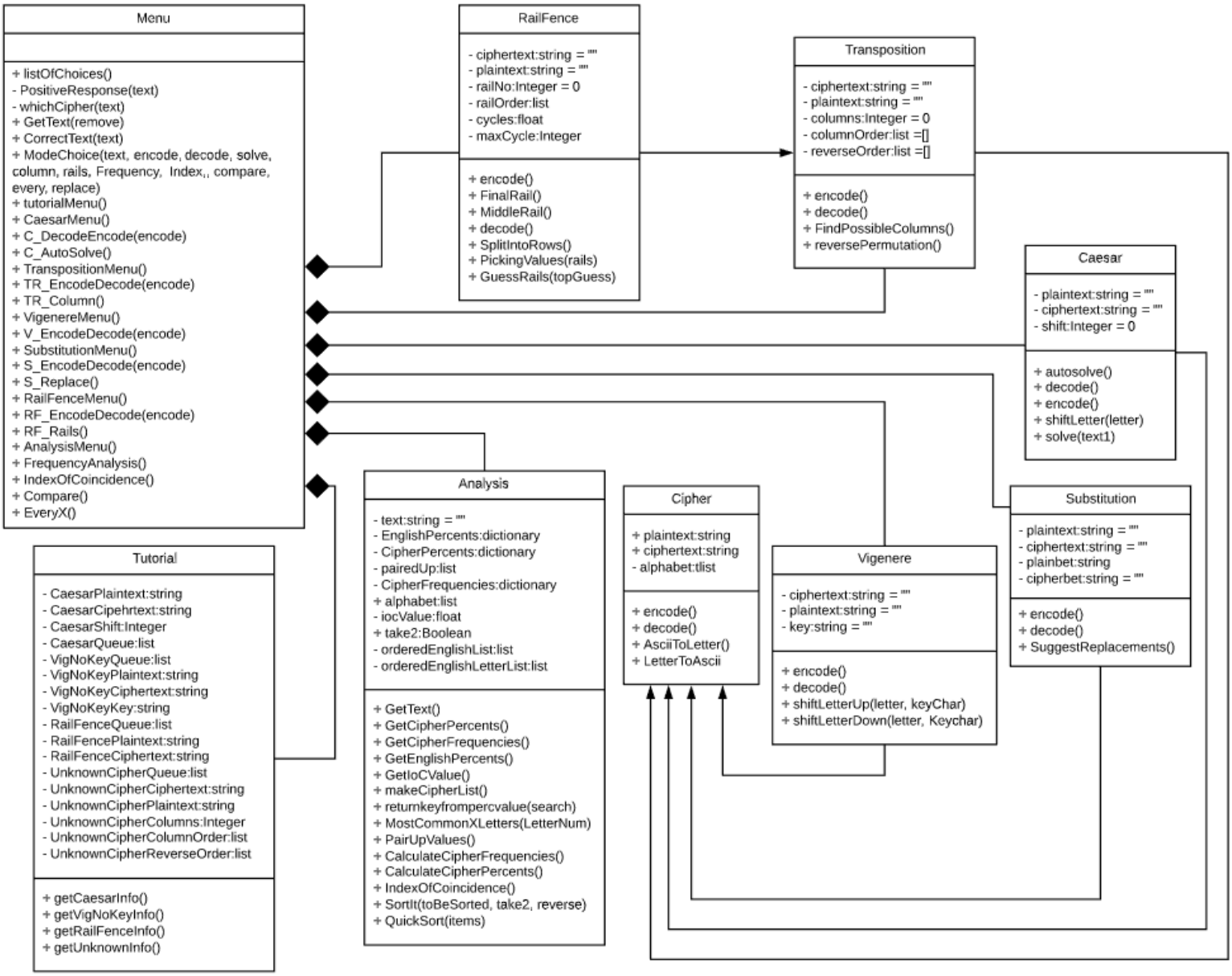
In my research before this project, I loved the idea of doing an assistant, or 'bot' to help with simple tasks, and it's genuinely nice to be able to apply it here. Due to my lack of knowledge and skills in actual natural language processing, including using machine learning to build up confidence in the nature of the response, I won't be able to create a perfect AI assistant, however this is acceptable for this project.

Variant Menu Choices

One of the main things about this style of approaching the project is the variance in responses that can come from the user, and still be accepted. This requires a lot validation and making sure that the user's intent is clear.
This variance in response will require me to allow multiple acceptable inputs, so there will have to be specific methods devoted the correct allowance of responses.

Classes

When I first assessed this project, there are a few distinct lines where I can create individual and encapsulated programs. This is mainly down to individual ciphers, and especially creating an entire analysis-based class would help to keep all the information processing in one area. Originally I thought about having separate classes for each process of the ciphers (one for encoding, one for decoding) however it was a lot easier keeping all the Vigenère information in one place rather than in 5, as there are potentially shared functions and capabilities across the same cipher. For example, in Caesar, you use the same algorithm but with a different shift number, so it is preferable if the encode and decode functions can access the same methods.



As you can see in this Class Diagram, it shows the relationships between all the classes. The main class, Menu, has a compositional relationship with the others, using them to process the user inputs. There is also shown the inheritance relationships with Cipher. The class Menu deals with reading in text from files or input, and ensuring they are in the right format for the ciphers to process them. It also holds the main personality and feedback to the user in the form of ADA. This is my personality or helper in the code, which stands for Advanced Decryption Assistant (and is in reference to the mother of Computer Science, Ada Lovelace).

Here are some sample data structures for the program. It shows that we'll have a selection of arrays (and array implemented structures), and be dealing with multiple dictionaries, especially in the Analysis class where the quick access of values is key. It also helps by identifying how these will be manipulated, which in the case of the Abstract Data Types (where Python doesn't have support for them) is extremely useful for implementation later.

Data Structure	**Example**	**Description**	**Manipulation**
Arrays/Lists	ReverseOrder	An array containing the order to decode a transposition. Calculated by reversePermutation or input by the user	Arrays are supported in Python, but this will be manipulated mainly through looping structures
	railorder	An array containing the order of rails for the program to run through	It's used in looping structures, and a pointer may be used to access it
	orderedEnglishLetterList	An array holding the letters of the English value, sorted high to low from their percentages	This is used in conjunction with the CipherPercents dictionary, and it will be accessed through indexing (so pointers will be necessary)
Queue	VigNoKeyQueue	A queue holding the next information for the tutorial. This stores the information and rubric for the Tutorial about a Vigenère decoding without a key.	This is used as if it is a static Queue that has already been filled. You take from the top and use pointers to access information.
Dictionary	CipherFrequencies	A dictionary holding the frequencies of each letter in the ciphertext. The key is the letter, e.g. 'a', and the value is the number of occurrences, e.g. 105.	Because Python deals well with dictionaries, it is being manipulated using the key, value pairs, with the key being the pointer to the value of the frequency.
	EnglishPercents	A dictionary holding the percentages of each of the letters, but in a standard or representative piece of text. This does not change in runtime, however it is used in the program.	Remarkably similar to Cipher Frequencies in manipulation. Accessed via the key (a-z) and the values are retrieved.

Another important piece of information is what the data is going to be like and how we are going to need to validate for it. You can see this in my Data Dictionary, which gives examples of a selection of variables and how the code can validate them.

Name	Purpose	Type	Example Data	Validation
ciphertext	To store the text to be decrypted, or has been encrypted	String	“VNVJKWTEZMLTZNYTK”	Not empty or whitespace
plaintext	To store the text to be encrypted, or has been decrypted	String	“HELLOWORLD”	Not empty or whitespace
CipherPercents	To store the percentages of each letter in the text, in order to aid code-breaking.	Dictionary	{"a": 8.04, "b": 1.48, "c": 3.34,"d": 3.82… }	Only changed by code, not entered
pairedUp	To store the pairs of letters that are calculated by using the two percentage values	Array	[[“a”, “e”], [“x”, “t”]…..]	Only changed by code, not entered
iocValue	To store the calculated IOC value (from the text).	Float	1.73	Only changed by code, not entered
up	A check in the RailFence class to understand which direction on the rails they currently are.	Boolean	False	Only changed by code, not entered
columnCorrect	Used as validation to make sure input is correct	Boolean	True	Only changed by code, not entered
columnIdeas	To store the suggest column numbers in the transposition menu.	Array	[1,2,5,10]	Only changed by code, not entered

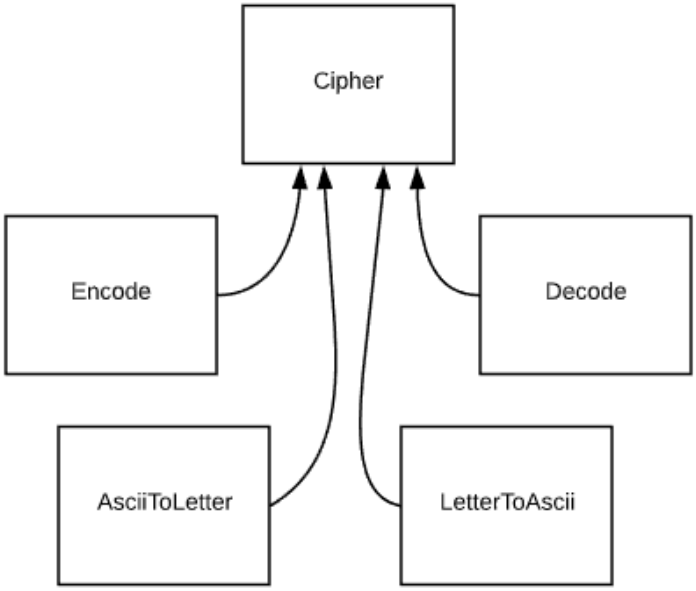
Cipher

Description & Explanation

This class is here to be used as an Abstract Base Class. Whilst Python doesn't have an implementation of interfaces and Abstract Classes, I have used the concept here in my project. What this is, is a class that is not directly called at any point in time however only inherited from. This provides a solid base for all the other classes, and it also allows some code to be shared by other Ciphers (for example, Vigenère and Caesar both use LetterToAscii as they have inherited it from Cipher).

Hierarchy Chart

Even though this class is not implemented, it still has some 'skeleton' functions. These are used by the other cipher classes or are placeholders for the other cipher classes to override.



Class Definition

Class Name:		Cipher	
Methods			
	Name		**Comments**
Public	Encode		It's a blank function to use as a template in an abstract base class
	Decode		Same as Encode
	AsciiToLetter		Used to convert a given value into it's ASCII character
	LetterToAscii		Converts the given letter into the ASCII value
Attributes			
	Name	**Data Type**	**Comments**
Private	alphabet	Array	It's an array with every letter of the English alphabet in it (a-z) in lowercase form
Public	plaintext	String	It's an empty string to use as a template in the abstract base class
	ciphertext	String	Same as plaintext

Caesar

Description & Explanation

A Caesar cipher is the easiest of the ciphers on this project. You simply shift each letter by a given value, using ASCII values. It's quite simplistic because of the ASCII shortcut, giving the programmer an easier way of encoding each letter into a specific number and back again. Due to this, I'll only be putting the Decode function in the Algorithms section as they are so similar.

The AutoSolve is slightly more complicated but still simplistic. Because Caesar has a period of 26, meaning it only has 26 possible states, the user could try every one individually, or rather just the code tries numbers 1—26 to see if the correct answer appears. It's not complex from a programming side, but very useful to a codebreaker, and you'll see it below.

Algorithms

Here are a few of the Caesar algorithms, as they are all quite simplistic. The most interesting things here are the shiftLetter (as it converts to and from ASCII) and the autoSolve which decodes using all the possible shifts.

Caesar Decode

```
SUBROUTINE decode(shift, ciphertext):
    shift *= -1
    plaintext = solve(ciphertext)
    RETURN plaintext
END SUBROUTINE

SUBROUTINE solve(shift, text1)
    text2 = ""
    FOR letter IN text1
      IF letter IS alpha THEN
        text2 += shiftLetter(letter, shift)
      ELSE
        text2 += letter
      END IF
    END FOR
    RETURN text2
END SUBROUTINE

SUBROUTINE shiftLetter(letter, shift)
    value = LetterToAscii(letter) + shift
```

Makes the shift negative

Calls the subroutine Solve

For every letter in the text, it checks if it is part of the alphabet, and shift it if it is.

The Alpha keyword is a check which checks if it's in the array ['a'… 'z' … 'A' … 'Z']

If it's not a letter it ignores it

Returns the text once complete

Adds the two values together

```
    IF letter  IS uppercase THEN
        value -= LetterToAscii("A")
        value = value MOD 26
        value += LetterToAscii("A")
    ELSE IF letter IS lowercase THEN
        value -= LetterToAscii("a")
        value = value MOD 26
        value += LetterToAscii("a")
    END IF
    newLetter = AsciiToLetter(value)
    RETURN newLetter
END SUBROUTINE
```

If the letter is uppercase ('A'-'Z'), then it uses the ASCII values for an uppercase letter, and corrects the letter back into the alphabet range

Does the same if lowercase ('a'-'z'), but using the lowercase ASCII values

After all of this, converts the value back into a character and returns it

Caesar AutoSolve

```
SUBROUTINE autoSolve(text):
    texts = []
    FOR x=0 TO 26
        shift = x
        plaintext = decode(text, shift)
        texts.append(plaintext)
    END FOR
    RETURN texts
END SUBROUTINE
```

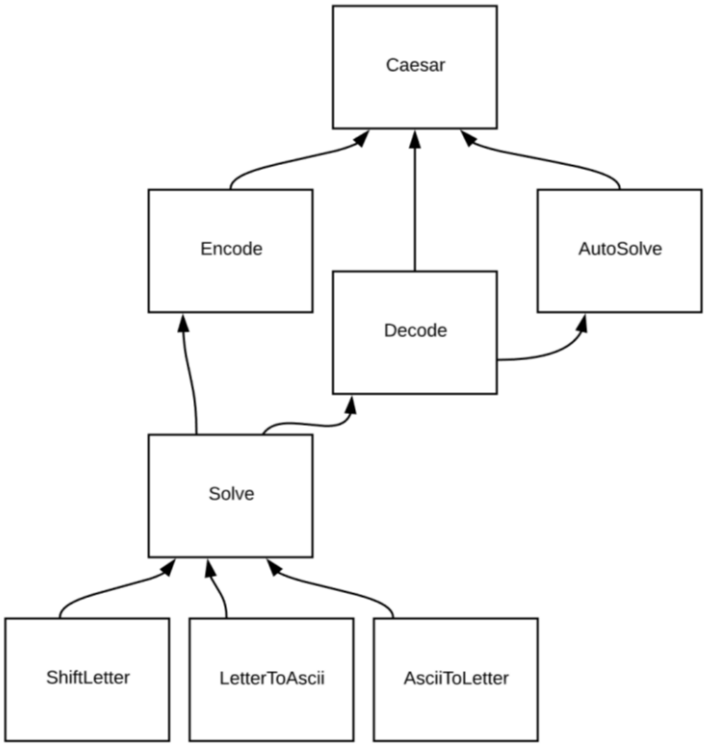
Loops through all the potential shifts (0 to 26)

Decodes using each of the shifts

Returns an array containing all of the potential texts

Hierarchy Chart

There are some simple relationships in the Caesar, as most of the functions depend on the one function of solve, as it does the 'heavy-lifting' of all the number crunching.



Class Definition

Class Name:	Caesar		
Methods			
	Name		**Comments**
Public	Encode		A high-level function that uses solve to add on the shift value
	solve		Used to add/remove shift value
	shiftLetter		Shifts each individual letter by the value
	decode		A high-level function that uses solve to remove the shift value
	autoSolve		Cycles through each potential shift value and creates an array with all the solved listed.
Attributes			
	Name	**Data Type**	**Comments**
Private	Plaintext	String	This stores the normal text, before it has been encoded or after it has been decoded
	ciphertext	String	This stores the encrypted text, before it has been decoded or after it's been decoded
	shift	Integer	This stores the amount (if specified) that the letters are to be shifted by

Substitution

Description & Explanation

This cipher seems overly complicated on paper, however there are some simple ways to make this cipher a lot less difficult. Initially I did some research into the way Python handles strings and found that Python has an inbuilt 'mesh' function. It allows you to give it a set of inputs, with a matched set of outputs, and it will translate the string from this instruction. Through some testing, this was simply the fastest way of doing this, as any method that I suggested had non-computable time (it was larger than a polynomial time complexity). This made me adapt this class to work with the in-built library function, called translate (and its sister, maketrans, which creates the 'mesh' for it to work with).

This does mean there is less design as a part of this class in the encoding and decoding aspect, however the most complex and interesting piece here is the SuggestReplacements, or my attempt at doing a Substitution AutoSolver. This uses the frequency analysis (from the Analysis class) to compare the percentage values of each letter to a piece of English text. Using the Analysis class, most of the complexity is hidden in there due to the nature of a compositional relationship, however I'll explain it here. You make a list of all the values, then using the frequencies of the letters (using the dictionary CipherPercents to be exact), you can pair them up using the normal English values, also stored as constants in the Analysis class.

Algorithms

Encode

```
SUBROUTINE encode(self, plaintext, cipherbet):
    ciphertext = plaintext.TRANSLATE(plaintext.MAKETRANS("a…z..A..Z", cipherbet))
    RETURN ciphertext
END SUBROUTINE
```

Uses the in-built function of Translate and Maketrans

SuggestReplacements

```
SUBROUTINE SuggestReplacements(ciphertext):
    suggestedList = Analysis().PairUpValues(ciphertext)
    RETURN suggestedList
END SUBROUTINE
```

Calls the analysis class in order to find the pairs

For Reference (from Analysis)

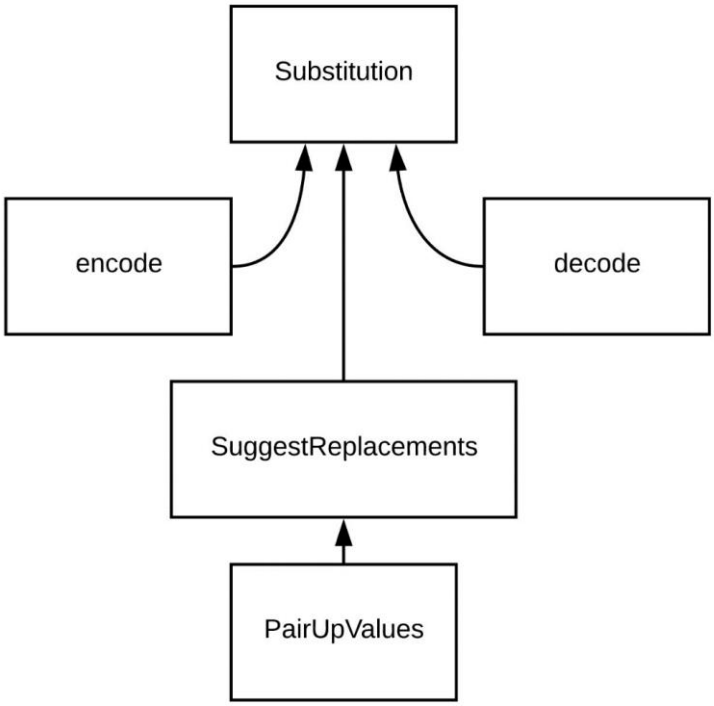
```
SUBROUTINE PairUpValues(ciphertext)
    cipherList = makeCipherList(ciphertext)
    FOR x=0 TO LENGTH(cipherList)
        value = cipherList[x]
        letter = returnkeyfrompercvalue(value)
        pairedUp.append([orderedEnglishLetterList[x], letter])
    END FOR
    RETURN pairedUp
END SUBROUTINE
```

Makes a list of all the percentages in the ciphertext

Pairs up the percentages with the percentages of the English text and takes the respective letters

Hierarchy Chart

Due to the simplistic nature of this class in general, there is no surprise that the Hierarchy chart is also quite simple. There aren't too many connections, and the 'PairUpValues' is a function that is from the Analysis Class.



Class Definition

Class Name:	Substitution	
Methods		
	Name	**Comments**
Public	encode	A function that uses a library method of translate and maketrans to substitute in specific values.
	decode	The same as encode, however it swaps the variables
	SuggestReplacements	This uses the Analysis class to find the paired up letters, and returns this is an array.
Attributes		
	Name	**Data Type**	**Comments**
Private	plaintext	String	This stores the text before it is to be encoded or after it has been decoded
	ciphertext	String	This stores the text before it has been decoded or after it has been encoded
	plainbet	String	The 'alphabet' which is the normal values (so a-z)
	cipherbet	String	The inputted 'alphabet' which are to be replaced/substituted

Vigenère

Description & Explanation

The Vigenère cipher is unique to spot, with its tell-tale 'flat distribution' in the frequency analysis. This is because each letter is changed by a different amount, as its key is a rolling string, repeated throughout the whole plaintext (or ciphertext). Apart from the fact that the letter is now the shift, it is just a set of Caesar ciphers, so once you know how to decode the Caesar the algorithm is pretty similar. In the algorithm section below, I will simply put the encode (& related ShiftLetterUp) because the encode and decode functions are so similar.

For the encoding algorithm, you've got to make sure that you don't try and change any punctuation, and if you miss out a space (because of the fact it isn't a letter) the key doesn't skip either. I've used the idea of trying the value outside the range, and if I don't get the error, I go ahead, but if I do get an error, then it just moves on and correct back to the front of the key. It's an interesting way to use the try/except structure in python.

ShiftLetterUp (and -Down) both work in a comparable way to shiftLetter in the Caesar class (above). Depending on the case of the letter they either use the uppercase ASCII value or the lowercase ones, and you proceed like normal from there! They use an inherited function from Cipher, the Abstract Base Class (ABC), which can convert the ascii into letter and letters into ascii, and this is especially useful because these classes are static methods. This means you do not need access to any of the normal attributes in a class (that you call using 'self.'), and by writing '@staticmethod' above the function (and not putting self into the parameters) then you end up with a normal function, except its in the class. This is particularly useful in this case because all this is doing is a simple conversion, so there is no need for access to the object & class variables.

Algorithms

Vigenère Encode

```
SUBROUTINE encode(plaintext, key):
    ciphertext = ""
    x = 0
    FOR letter IN plaintext:
        IF letter IS alpha THEN
            newLetter = shiftLetterUp(letter, key[x])
            TRY:
                key[x+1]
                x += 1
            EXCEPT:
                x = 0
            ciphertext = ciphertext + newLetter
        ELSE
            ciphertext = ciphertext + letter
```

Initialises variables

Loops through each letter in the plaintext

If it's a letter (the keyword 'alpha' mean is it is in the range 'a'-'z' or 'A'-'Z') then you shift it, if not ignore that character

Using the try/except structure, it tests to see if it errors, if it doesn't then you move the key along, if It does error, then you revert to the beginning

```
        END IF
    END FOR
    RETURN ciphertext
END SUBROUTINE
```

Returns the completed ciphertext

```
SUBROUTINE shiftLetterUp(letter, keyChar):
    IF letter IS lowercase THEN
        shifted = AsciiToLetter((((LetterToAscii(letter)-97) + (LetterToAscii(keyChar)-97)) %26) + 97)
    ELSE IF letter IS uppercase THEN
        shifted = AsciiToLetter((((LetterToAscii(letter)-65) + (LetterToAscii(keyChar)-65)) %26) + 65)
    END IF
    RETURN shifted
END SUBROUTINE
```

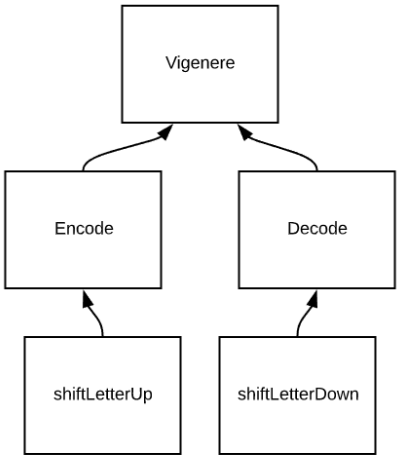
Uses the lowercase ASCII values

Uses the uppercase ASCII value

Returns the letter

Hierarchy Chart

There aren't too many connections on this hierarchy chart, but the functions themselves aren't too simple. The encode (or decode) keep track of where you are in the key and send the two letters int shiftLetterUp (or shiftLetterDown).



Class Definitions

Class Name:	Vigenère	
Methods		
	Name	**Comments**
Public	encode	A function that uses shiftLetterUp, goes through every letter and finds the next key letter
	shiftLetterUp	Shift each letter by the current key letter, by adding it on
	shiftLetterDown	Shift each letter by the current key letter, by removing the value instead
	decode	A function that uses shiftLetterDown, goes through every letter and finds the next key letter
Attributes		
	Name	**Data Type**	**Comments**
Private	ciphertext	String	This stores the text before it has been decoded or after it has been encoded
	plaintext	String	This stores the text before it is to be encoded or after it has been decoded
	key	String	This is a string to use to shift each letter by

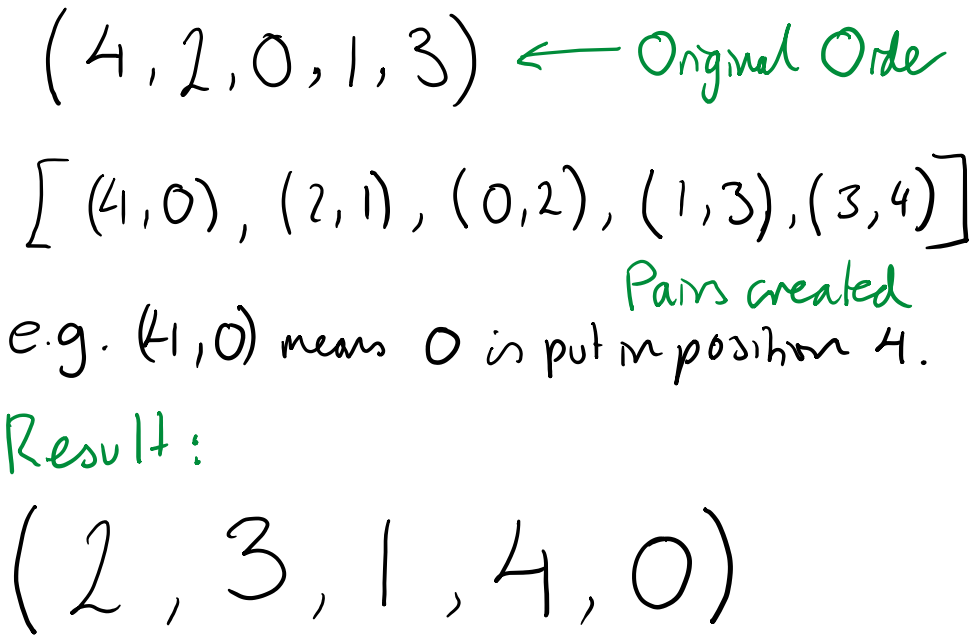
Transposition

Description & Explanation

A Transposition ciphertext is really interesting to work with, as you can always put it into 2 potential formats, however in a 'shuffled' transposition with an order for the columns, there is only one way will which start to spit out a solution which looks correctly deciphered.

One of the most interesting things about the Transposition when it is 'shuffled' or encrypted with a key (the order of columns) is that if you have the original order of the columns, you have to perform a decryption of the key as well!

Here I have called this function ReversePermutation, as you have to find the reverse of the key. What you need to do with this is for every value in the key, you create a pair of the value and the position. Then the position number goes in the position from the value. To explain it better, here is a diagram explaining it:



The rest of the decoding is simply picking the correct values based upon this new key, which is very similar to the encoding of Transposition. In the Algorithms section I have only put Decode, ReversePermutation and FindPossibleColumns, as Encode is so similar in implementation as Decode.

Algorithms

Transposition Decode

```
SUBROUTINE decode(ciphertext, columns, columnOrder)
        Reversed = ReverseColumns(columnOrder)
        rows = LENGTH(ciphertext) ÷ columns
        FOR x = 0, TO rows
                FOR i = 0 TO LENGTH(Reversed)
                        Index = Reversed[i]
                        words += ciphertext[(index × rows) + x]
                END FOR
        END FOR
        RETURN words
END SUBROUTINE
```

Finds the reverse of the key

Skips through the text by every index*rows, in order to pick the correct letter

It returns the final string

```
SUBROUTINE ReverseColumns(columnOrder)
        FOR i = 0 TO LENGTH(columnOrder)
                Item = ColumnOrder[i]
                index.append((Item, i))
        END FOR

        reverse = []
        FOR i = 0 TO LENGTH(Index)
                reverse.append(0)
        END FOR

        FOR x = 0 TO LENGTH(Index)
                elem = index[x]
                reverse [elem[0]] = elem[1]
        END FOR

        RETURN reverse
END SUBROUTINE
```

Creates a pairing of position and value in the form (value, position)

Creates an empty array the right size in order to place the values in the right positions

Let the Index pair be (key, value). It puts the value in the position which is numbered key

Returns the reversed array

FindPossibleColumns

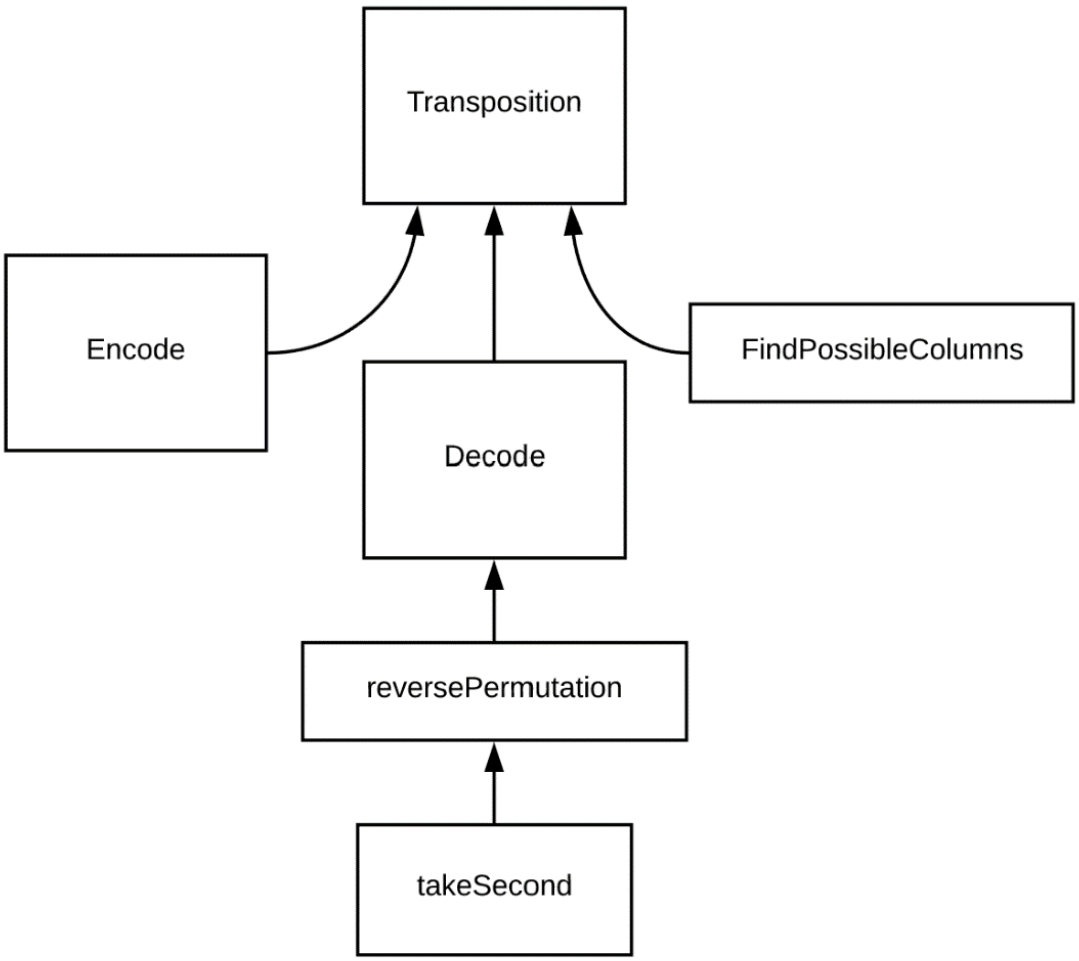
```
SUBROUTINE FindPossibleColumns(ciphertext):
    factors = []
    FOR i=1 TO LENGTH(ciphertext)) + 1
      IF LENGTH(ciphertext) % i == 0 THEN
        factors.append(i)
      END IF
    END FOR
    RETURN factors
END SUBROUTINE
```

This is a simple factorisation algorithm, which runs through all values and checks to see if there is a factor. This is because the transposition is encoded by having a full grid, and therefore it must be factorizable

Returns an array of the potential values

Hierarchy Chart

This is showing the relationships of the class. You can see that Decode requires a few more connections, whereas Encode and FindPossibleColumns are a little simpler.



Class Definition

Class Name:	Transposition	
Methods		
	Name	**Comments**
Public	encode	This uses the columnOrder to jump around the text to pick the correct letters.
	decode	This uses the reverseOrder to jump around and take the correct letters
	FindPossibleColumns	This uses the factors of the text to figure out how the text might fit neatly into the grid specified
	reversePermutation	This function calculates the reverse of the key, by for every item, it becomes its position
	takeSecond	This simply returns the second item in the item inputted (so works for an array, tuple, etc)
Attributes		
	Name	**Data Type**	**Comments**
Private	ciphertext	String	This stores the text before it has been decoded or after it has been encoded
	plaintext	String	This stores the text before it is to be encoded or after it has been decoded
	columns	Integer	This is the number of columns
	columnOrder	Array	All the numbers 0 to columns, in a user specified order
	reverseOrder	Array	The inverse of the ColumnOrder (as specified by the method reversePermutation), or user specified.

Rail Fence

Description & Explanation

This class deals with the RailFence (or ZigZag) cipher. This is a really difficult cipher to understand and follow, especially because you have to keep track of so much information, especially when decoding it, to understand where you are and what to do next.

I split this into 3 "sections". An encode, a decode, and a function where you can 'guess' the number of rails for an encoded rail fence (or find how many rails you can encode it with when you're decoding). The encode section was the most complex as you have the algebra to deal with here. Each rail has a different encoding system, and because I am doing this for any number of rails, I needed a system that can cope with both odd and even rails.

After some calculation, I figured out that the top rail takes every 2*railNo -1 (if you index from 1). I had to work this correctly so that it picked the right letters. I also figured out the algebra for the middle rail, which is just two letters, but either side of the current rail. The final rail is the easiest, which is just every railNo (if indexing from 1). If you would like to see the full encryption algorithm, it is just below, under 'Algorithms'.

To decode the railfence, you need to know the order that the rails are going to be in, and therefore which of them to take. To do this you need the rails each individually separated and in the correct order. You also need to keep track of whether you are going up or down them, for this I've ended up using a Boolean. You can see the full decryption algorithm just below.

Algorithms

Rail Fence Encode

```
SUBROUTINE encode(railNo, MaxCycle, plaintext)
        ciphertext += FirstRail(cycles, plaintext, railNo)
        currentrail =1

        WHILE currentrail != (railNo-1)

                ciphertext += MiddleRail(currentRail, maxCycle-currentrail, plaintext, maxCycle)
                currentrail +=1

        END WHILE

        ciphertext += FinalRail(plaintext, maxCycle, RailNo)

        RETURN ciphertext

END SUBROUTINE

SUBROUTINE FirstRail(cycles, plaintext, railNo)

        words = ""

        FOR x = 0 TO (cycles+1)

                words += plaintext[x × ((2 × railNo) − 2)]

        END FOR

        RETURN words
```

Calls the functions FirstRail, then middleRail railNo-2 times

MiddleRail takes the inputs currentRail and MaxCycle-currentRail, as this takes the two opposite ends of the 'zigzag'

To end, finally calls FinalRail

You repeat cycles+1 times as the number of character on the top rail is c+1

Takes every 2*RailNo -2 letter, as it has to jump the down and up section of the zigzag

```
END SUBROUTINE

SUBROUTINE MiddleRail(first, second, plaintext, maxCycle)
        words = ""
        FOR x = 0 TO LENGTH(plaintext) INCREMENT maxCycle
                words += plaintext[x + first]
                words += plaintext[x + second]
        END FOR
        RETURN words
END SUBROUTINE

SUBROUTINE FinalRail(plaintext, maxCycle, RailNo)
        words = ""
        FOR x = 0 TO LENGTH(plaintext) INCREMENT maxCycle
                words += plaintext[x + (RailNo - 1)]
        END FOR
        RETURN words
END SUBROUTINE
```

It increments by maxCycle so the text makes sure to jump though the middle of the ZigZags

Does two at a time because there is an up and down value on each middle rail

Takes every railNo-1 as the last element is always the bottom of the cycle, and that occurs every railno-1 (if you start from 0)

Rail Fence Decode

```
SUBROUTINE decode(text, railno)
    rails = SplitIntoRows(text, railno)
    text = PickingValues(rails)
    RETURN text
END SUBROUTINE

SUBROUTINE SplitIntoRows(text, railno):
    toprail = ""
    bottomrail = ""
    rails = []
    counter = -1
    n = (len(text)-1)//((2* railno)-2)
    TRY
      FOR x =0 TO n+1
        counter +=1
```

Calls two subroutines (one to split into the rails and the other to pick the correct values from each) and returns the text

Initialises variables

An error catching statement

Takes the number of letters for the first row

```
        toprail += text[counter]
    END FOR
    rails.append(toprail)
    FOR i=0 TO railNo-2
        temp = ""
        FOR x TO n *2
            counter +=1
            temp+= text[counter]
        END FOR
        rails.append(temp)
    END FOR
    FOR x TO n
        counter +=1
        bottomrail += text[counter]
    END FOR
    rails.append(bottomrail)
  EXCEPT
    PASS
  RETURN rails
END SUBROUTINE
SUBROUTINE PickingValues(rails)
  current = 0
  up = True
  text = ""
  WHILE rails[0] <> "" DO
    listOfChars = rails[current]
    first = listOfChars[0]
    fixed = listOfChars[1:]
    rails[current] = fixed
    text+= first
    IF up = TRUE THEN
```

Takes the middle rows (double the number as up and down) but railNo-2 times in order to do all the middle rails

Finally takes the last row

If there has been an error, ignore it and end, but don't error

Return the array containing all the rows separately

Initialises variables

While we haven't exhausted the top rail

Remove the first element of the current rail and replace back into the 2D array rails

Add the letter onto the decoded text string

```
        current+=1
    ELSE
        current-=1
    END IF
    IF current>= len(rails) THEN
        current -=2
        up = FALSE
    ELSE IF current <0 THEN
        current+=2
        up = TRUE
    END IF
  END WHILE
  RETURN text
END SUBROUTINE
```

Now you begin to look at changing rails. You start by just changing it.

You now do a check if it's over the end of the rails, and change the Boolean & correct the current back onto the right range

Once you've reached the end of the while loop, return the text

Guess Rails

```
SUBROUTINE GuessRails(text, topGuess)
  potentials = []
  FOR x=2 TO 1+topGuess
    IF (len(text) MOD ( (2*x) -2)) == 1 THEN
      potentials.append(x)
    END IF
  END FOR
  RETURN potentials
END SUBROUTINE
```

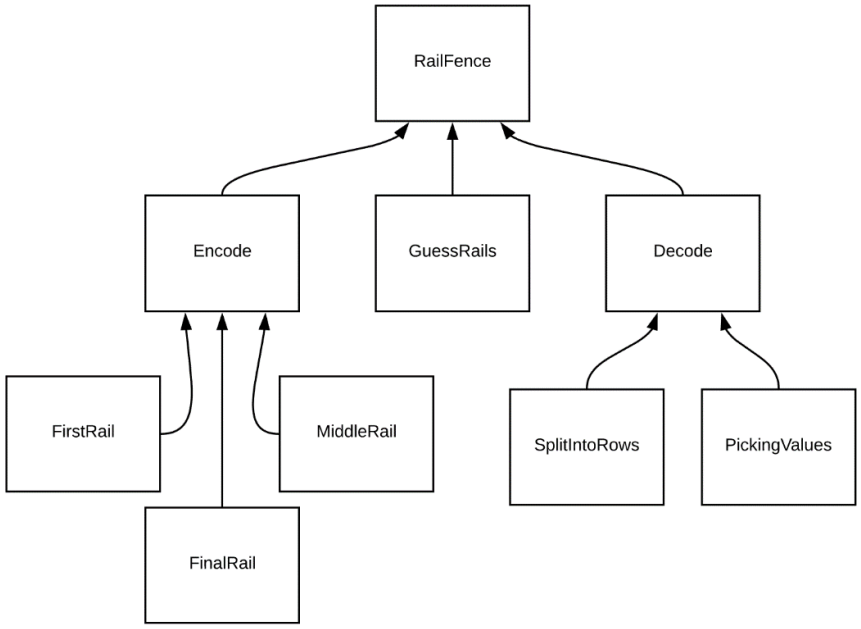
Initialises the variable

Loops through every number 2 to Top Guess

This statement is calculated by the fact the text must have a multiple of the MaxCycle, but plus one for the last character. If it is suitable it adds it onto the array.

Returns the array and ends

Hierarchy Chart



Here is the hierarchy chart for the rail fence class. You can see how individual functions will link into one another. GuessRails is an alone function, whereas for encode and decode they need a lot of inputs to correctly function.

Class Definition

Class Name:	RailFence	
Methods		
	Name	**Comments**
Public	encode	This calls the methods FirstRail, MiddleRail, FinalRail in order to take the correct letters in the right order. It calls MiddleRail railNo-2 times.
	FinalRail	It takes every railNo-1 letter, jumping up by the maximum cycles
	MiddleRail	Using the two integers input, it takes the next two letters (the 'up' letter and the 'down' letter)
	FirstRail	Takes every letter multiplied by2*RailNo -2, until it's reached the end of the list
	decode	This calls SplitIntoRows, and puts the information into PickingValues.
	SplitIntoRows	It splits the text into the First rail, railno-2 Middle rails, and the final rail. Then returns these as an appended array, with each element being a different row.
	PickingValues	Using the array from SplitIntoRows, it takes a letter from each array, but once it's reached the bottom, knows to reverse the movement – using the Boolean local variable up. Returns the completely decoded railfence
	GuessRails	Using the fact that the modulus of the length of text by 2*rails -2 should be one complete cycle, ciphertext MOD 2*rails -2 should equal 1, because of the extra letter on the first rail. This runs

			through all potential numbers (entered into the method via the parameter topGuess), and returns an array of all the numbers that is true by this logic.
Attributes			
	Name	**Data Type**	**Comments**
Private	ciphertext	String	This stores the text before it has been decoded or after it has been encoded
	plaintext	String	This stores the text before it is to be encoded or after it has been decoded
	railNo	Integer	The number of rails for the railfence to be put on.
	railOrder	Array	Generated by the code. An array going down then up the rails. E.g. [0,1,2,3,2,1]
	cycles	Float	The number of cycles in the code from the length of the plaintext. If a bad railNo, it will be a float.
	maxCycle	Integer	The maximum number of letters in the cycle (which is down then up the rails)

Analysis

Description & Explanation

When you are given an unknown cipher as a part of the cipher challenge, then you initially try different methods of analysis to point out key characteristics. This is what this class is mainly used for – all the specifics and in-depth analysis of the text. From the Index of Coincidence to calculates the frequency analysis, the Analysis class is most used when approaching a completely new cipher. There is a massive amount of complexity here, especially when we look at particular subroutines such as SortIt and CalculateCipherFrequencies, and I've drawn in some different ideas here to speed up the analysis process.

One key feature in CalculateCipherFrequencies is the functional programming methods I have used. Before this project, I had a little experience in Haskell, a functional programming language, and knew that it's extremely quick number-crunching way of approaching programs would be extremely useful here, as I have to deal with a large amount of data (in my analysis we found that an average text size of the competition was anywhere between 2,000 to 3,000 characters, and simply running through a string this long will take a while, let alone the computation required for each of the letters. So in order to streamline the code, I decided to use the idea of a functional programming 'filter'. This uses two inputs, one of which is a function which returns a Boolean value, then other, something which can be passed through that function. I will use this to calculate the number of letters there are in the text of each type, as it quickly returns the items that pass the test (that return a value of True).

Also, with this, I need the function in the filter to take different inputs, as I want to be able to use this over all the letters in the alphabet. In my research I came across the idea of a Lambda function, which is used in both Maths and Functional Programming. In maths a function $f(x) = x^2$ can then be written as $f: \rightarrow \lambda x. x^2$. This can be applied in Haskell (the functional programming language) like $(\backslash x -> x^2)$, and in Python, the language of choice here, you use the keyword lambda, and evaluate like this: $(lambda\ x: x^2)$. I have used this idea when filtering my letters as the function in the filter needs to be able to take any letter a-z, and if the function has to be a specific one, it will cause more trouble in order to incorporate it.
The main reasons you use the lambda functions in functional Programming is when the function is only used once, and it streamlines the code a lot more. If I didn't have this 'flexible function', then there would have to be a way of changing the function in run time, however from my knowledge, Python does not support partial application, another functional programming idea. To see the full algorithm, I will put it below in the Key Algorithms section.

Also, in Analysis, I use the idea of a quicksort to sort my dictionaries (and other data structures) and return the highest letters and value pairs (if a dictionary). I didn't want to use the inbuilt sort in the code and found in my research the idea of a quick sort. It simply pivots repeatedly the items into above and below a certain value, then combines them all at the end. This is similar an idea to a merge sort, without too much complexity. In order for the quicksort function to work correctly, I will also implement a 'feeder' function called SortIt, which prepares the data structure to be used in the Quicksort, using a type check. This validation is extremely useful, otherwise the sort would not work correctly. To see this, look below in the Algorithms section.

Another interesting algorithm in the Analysis class is the calculation of the Index of Coincidence value. This uses the frequencies of each letter to calculate how random the text is and compare it to the usual randomness of a representative piece of English (which is 1.73). It's also below.

Key Algorithms

Some of the most interesting algorithms in my Analysis class use some functional programming concepts and the intense manipulation of dictionaries. There's also the calculation of the Index Of Coincidence, which tell the codebreaker whether their text is very random or not.

Quick Sort

```
SUBROUTINE SortIt(toBeSorted, take2 = False, reverse=False)
    listOfItems = []
    IF TYPE(toBeSorted) IS list THEN
        listOfItems = toBeSorted
    ELSE IF TYPE(toBeSorted) IS dict THEN
        FOR key IN toBeSorted
            listOfItems.append((key,toBeSorted[key]))
        END FOR
    END IF
    newList = QuickSort(listOfItems, take2)
    IF reverse == True THEN
        newList.reverse()
    END IF
    RETURN newList
```

This does a type check to make sure it is in a list or array style type

If it is a dictionary, it converts it into an array of (key,value) tuples

Now that the data is corrected, a quicksort can begin. The take2 value is for identification of the tuple case

The quicksort is ascending by nature, if the code needs a descending array, it reverses it

```
SUBROUTINE QuickSort(items, take2)
    IF LENGTH(items) == 1 THEN
        RETURN items
    ELSE IF items == [] THEN
        RETURN []
    ELSE
        LessList = []
        MoreList = []
        pivot = items.pop(0)
        FOR element IN items:
            IF take2 THEN
```

Because this is a recursive program/sort, you need your general and base cases. The code can potentially not have anything bigger, or only one item, so both of these base cases are needed

Initialises variables

For a quicksort you can use any item as the pivot, but I have chosen the first element

Loops through all the elements, and compares them to the first element

```
            IF element[1] < pivot[1] THEN
                LessList.append(element)
            ELSE
                MoreList.append(element)
            END IF
        ELSE
            IF element < pivot THEN
                LessList.append(element)
            ELSE
                MoreList.append(element)
            END IF
        END IF
    END FOR
    newLess = self.QuickSort(LessList)
    newMore = self.QuickSort(MoreList)
    newList = newLess + [pivot] + newMore
    RETURN newList
  END IF
END SUBROUTINE
```

The reason for the two sets of compairsons is if the code is dealing with a list of tuples rather than a standard array.

Now sorts the two lists separately, and recursively, then appends on the values and returns them up the stack

CalculateCipherFrequencies

```
SUBROUTINE CalculateCipherFrequencies(text)
    CipherFrequencies = DICT('a':0, 'b':0... 'z':0)
    FOR key IN CipherFrequencies
        number = LENGTH(FILTER((\x -> x = key), text)))
        CipherFrequencies[key] = number
    END FOR
    RETURN CipherFrequencies
END SUBROUTINE
```

A blank dictionary of letters a-z with a value of 0 is initialisec

For every letter, it find the length of the filtered text, when the letter is equal to the letter being looped

Stores back in the dictionary, and return the dictionary once finished

Index Of Coincidence

```
SUBROUTINE IndexOfCoincidence(text)
    cipherFreq = CalculateCipherFrequencies(text)
    F = 0
    phiO = 0
    FOR key IN CipherFrequencies
        F = (CipherFrequencies[key]) * (CipherFrequencies[key] -1)
        phiO += F
    END FOR
    N = LENGTH(text)
    phiR = N * (N-1) * 0.0385
    IF phiR > 0 THEN
        ioc = phiO / phiR
    ELSE
        ioc = None
    END IF
    RETURN ioc
END SUBROUTINE
```

Calls the function CalculateCipherFrequencies

For each of the letters, you multiply the frequency by the frequency -1

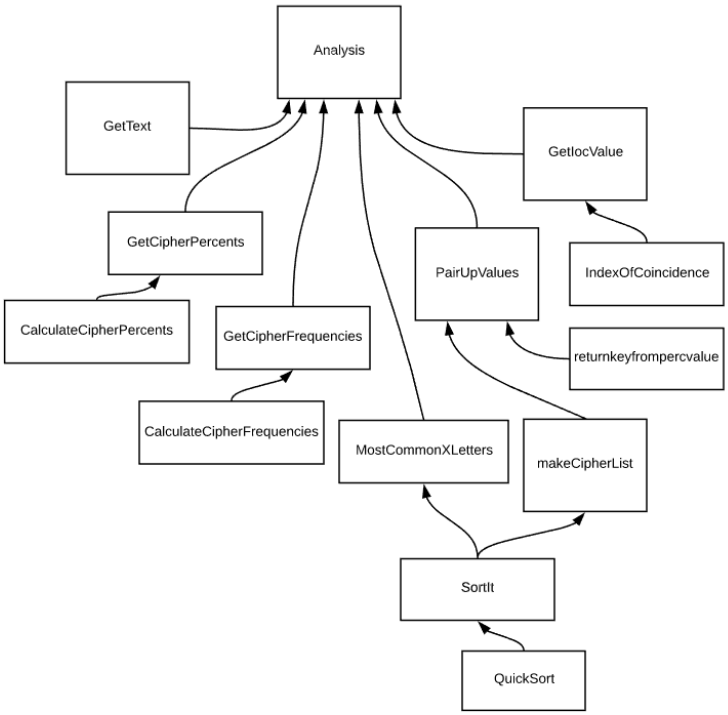
Then you add onto a running total

To find the next value, you multiply the length of the text by itself-1 and by the randomness constant, 0.0385

As you cannot do 0 division…

The Index of Coincidence value is returned

Hierarchy Chart



Class Definition

Class Name:	Analysis	
Methods		
	Name	**Comments**
Public	GetText	A getter returning the text after it has been corrected
	GetCipherPercents	A getter returning the dictionary CipherPercents
	GetCipherFrequencies	A getter returning the dictionary CipherFrequencies
	GetEnglishPercents	A getter returning the dictionary holding the English language letter percentages
	GetIOCValue	A getter returning the Index Of Coincidence value (a float)
	makecipherlist	It makes a list of the all values (from the key, value pairs) of the cipherPercents dictionary
	returnkeyfrompercvalue	This uses the dictionary CipherPercents to find the key for the value inputted
	MostCommonXLetters	Using the value input (LetterNum) it takes every combination of those letters and creates a frequency list of the number of occurrences.
	PairUpValues	Pairs up high English percents to high cipher percents. Used to figure out the potential matches in a substitution cipher
	FilterLetters	A simple function that checks if the letter is equal to the current letter on the program
	CalculateCipherFrequencies	Calculates the number of letters for the specific letter, and puts it into the dictionary

	CalculateCipherPercents	Uses the dictionary CipherFrequencies and calculates the percentage of those letters out of the length of the entire text.
	IndexOfCoincidence	Using the specific formula, it calculates a value which tells you how random the text is. By looking at this, you can identify the
	SortIt	This prepares the dictionary or array to be sorted by the function QuickSort
	QuickSort	This performs a quicksort on the array inputted. It recursively calls itself until the array is empty or has only one item left.

Attributes			
	Name	**Data Type**	**Comments**
Private	text	String	Stores the text, minus punctuation and all lowercase
	EnglishPercents	Dictionary	A dictionary that is a constant. It tstiores the usual percentages of letters in the English alphabet (from a representative piece of text).
	CipherPercents	Dictionary	A dictionary (comparable to EnglishPercents) which stores the percentages of letters but for this specific piece of text.
	pairedUp	Array	The ciphertext letters paired up to the normal English letters (based on percentage) in order to compare them.
	CipherFrequencies	Dictionary	Similar to CipherPercents, except it stores the occurences of each letter in the ciphertext.
	currentletter	String	This is a variable that is used to filter the letters
	iocValue	Float	A value calculated by IndexOfCoincidence, that represents how random the text is. Usually given to 2 decimal places, but the code gives it to any (no round involved)
	OrderedEnglishList	Array	A list of all the English (percentage) values in order (high -> low) because it's more time-consuming to resort the list every time.
	orderedEnglishLetterList	Array	Linked to the OrderedEnglishList, except it has the letters. This is again to not spend too much time resorting the list each time through.
Public	Take2	Boolean	It is a value that you use to identify whether you're using a tuple/2D array or a simple array in the function Quicksort as it has to be able to deal with either data types
	alphabet	Array	Filled with the letters a-z (lowercase) to be used in identification and loops (inside and out the code)

Tutorial

Description & Explanation

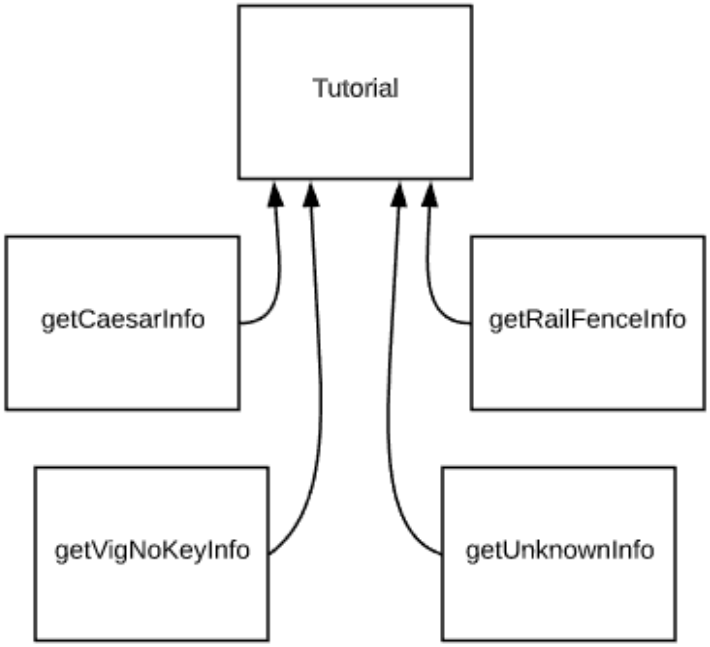
There are no complex algorithms in the Tutorial class, as it simply contains all the information for the tutorial, plus a few simple Getters to return the information the Menu or user if required. But to explain the reasoning for this is that the tutorial will have to have a ciphertext and plaintext to work from, plus the relevant information with it. This stores all this information and the program that then uses this can choose whether they need the plaintext or the shift (or whatever the information is). This class is especially useful to store in one place all the texts and information, and especially if you would potentially find a mistake in the text.

Algorithms

There are no key algorithms in this class.

Hierarchy Diagram

For this class, the hierarchy diagram is very simple. It's 4 getters, each for the specific cipher it is returning information on. Each are easily callable and simple to access.



Class Definition

Class Name:	Tutorial	
Methods		
	Name	**Comments**
Public	getCaesarInfo	A simple getter that returns the relevant information for the Caesar Tutorial
	getVigNoKeyInfo	A simple getter that returns the relevant information for the Vigenere (but without a key) Tutorial
	getRailFenceInfo	A simple getter that returns the relevant information for the RailFence Tutorial
	getUnknownInfo	A simple getter that returns the relevant information for the Unknown cipher Tutorial

Attributes			
	Name	**Data Type**	**Comments**
Private	CaesarPlaintext	String	The plaintext used in the Caesar tutorial
	CaesarCiphertext	String	The ciphertext used in the Caesar tutorial
	CaesarShift	Integer	The shift used in the Caesar tutorial
	CaesarQueue	Array	The queue that holds all the information for the Caesar tutorial, including into, outro and explanations with ciphertext as you go
	VigNoKeyQueue	Array	The queue that holds all the information for the Vigenère but without a key tutorial, including into, outro and explanations with ciphertext as you go
	VigNoKeyPlaintext	String	The plaintext used in the Vigenère (but without a key)'s tutorial
	VigNoKeyCiphertext	String	The ciphertext used in the Vigenère (but without a key) tutorial
	VigNoKeyKey	String	The key used to encrypt the Vigenère in the Vigenère tutorial
	RailFenceQueue	Array	The queue that holds all the information for the Rail Fence tutorial, including into, outro and explanations with ciphertext as you go
	RailFencePlaintext	String	The plaintext used in the Rail fence tutorial
	RailFenceCiphertext	String	The ciphertext used in the Rail Fence tutorial
	UnknownCipherQueue	Array	The queue that holds all the information for the Unknown cipher's tutorial, including into, outro and explanations with ciphertext or info/data as you go
	UnknownCipherCipherText	String	The ciphertext used in the Unknown cipher's tutorial
	UnknownCipherPlainText	String	The plaintext used in the Unknown Cipher's tutorial
	UnknownCipherColumns	Integer	The number of columns used to encrypt the Unknown cipher in the tutorial (it was a transposition)
	UnknownCipherColumnOrder	Array	The encoding order of the cipher in the Unknown cipher tutorial
	UnknownCipherReverseOrder	Array	The decoding order of the cipher in the Unknown cipher tutorial

Menu

Description & Explanation

This is the only non-cipher-based class; however, it is necessary to complete the project. This class ties together all the ciphers so far, including adding the 'Natural Language'-like features and ensures that the data input into each of the cipher classes is valid and provides the right feedback and information to the user. It makes the code usable, especially to beginners, and finally ties together all the pieces of this project in a neat way.

In this menu system, I have thought about implementing a personality, nicknamed ADA (see the Natural Language section for more). She will guide the users through the code, including giving the relevant error messages whilst still being friendly. From this, each different input for choices, as I am using some natural language ideas, has an array that stores all the potential responses that would constitute that option has been chosen.
For example, for the Analysis option, the user might input "analysis", "analyse", or "frequency analysis" or any similar phrase. I will put all the ones I can think of into the array, and these will provide the basis of my "natural language" features, as they can use any number of potential phrases. There will be a specific function, called WhichCipher, dealing with the selection of the ciphers, which returns set keywords that will trigger opening another menu, for example. There will also be a PositiveResponse function too, which is when the user is asked a yes/no question. This allows the code to quickly identify (with a Boolean) the intent of the response.

There also is required a large amount of validation and defensive programming. Especially for those ciphers which require some extremely specific information in defined boundaries, there will be intensive validation before the information is then put into the cipher classes. There will lots of text manipulation, and especially for the Substitution cipher, a confirmation that there are all the letters a-z in the alphabet, then the uppercase alphabet also added (due to the nature of the function). The analysis menu will have to print the information in an easy to read way, as it's a lot of numbers, which the user may need guiding through.

I have the idea of using 1 function to deal with both encoding and decoding (where possible) as they often need the same information, so instead of having, for example a Caesar Encode menu and a Caesar Decode menu, but a Caesar Encode/Decode Menu. I've put an exemplar in my algorithms section below, involving the Rail Fence Encode/Decode. However, for specific functions like Guessing the columns in a transposition will be separate. This does mean the Analysis menu will have a lot of bespoke functions, as it does a lot of quite different things.

Due to the fact the Tutorial class is simply some stored variables, the Tutorial menu will be used to cycle through them, adding some flavour-text hopefully and presenting the information in a neat and easy to understand way.

One key function I'd like to implement though is the idea of pauses frequently in the code, using time.sleep(x). This will slow the code down, but in such a way that is aims to look like a user is typing. I really like this idea as it makes ADA feel a little more human when she's not so inhumanly fast at typing. I will have to fine-tune the pauses though as it may feel unnatural. This would be especially key in the tutorial section, as it gives the reader some time to read each new statement as it comes in.

Algorithms

Tutorial Menu

```
SUBROUTINE tutorialMenu()
    OUTPUT “ADA: Welcome to the Tutorial!"
    OUTPUT "ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.\nADA: For very beginners, I recommend our Caesar or RailFence tutorials. \nFor more advanced codebreakers, the best place to start would be the Vigenere (which uses a crib) and the Unknown Cipher."
    tutorialchoice = INPUT "ADA: Which cipher would you like to see a tutorial of? \nYou: "
    cipherChosen = whichCipher(tutorialchoice)
    IF cipherChosen == "C" THEN
       queue,  plaintext, ciphertext = Tutorial().getCaesarInfo()
    ELSE IF cipherChosen == "RF" THEN
       queue,  plaintext, ciphertext = Tutorial().getRailFenceInfo()
    ELSE IF cipherChosen == "U" THEN
       queue,  plaintext, ciphertext = Tutorial().getUnknownInfo()
    ELSE IF cipherChosen == "V" THEN
       queue,  plaintext, ciphertext = Tutorial().getVigNoKeyInfo()
    ELSE
       OUTPUT "ADA: That doesn't appear to be a cipher in my databanks. Could you try another?"
    END IF
    OUTPUT “ADA: Now, onto the tutorial!"
    WAIT(3)
    Intro = POP(queue)
    OUTPUT "ADA: " + str(Intro)
    WAIT(3)
    FOR x=0 TO LENGTH(queue)-1
       pairOfInfo = queue[x]
       OUTPUT "ADA: " + str(pairOfInfo[0])
       WAIT(3)
       OUTPUT "ADA: " + str(pairOfInfo[1])
       WAIT(3)
    END FOR
```

It has an intro message

Takes an input and checks to see if that cipher is one of the few available

Once the specific cipher has been loaded in, it takes the first element off the queue

For all the middle section, which is the decoding and the ciphertext as it goes, it repeats until it is done

```
    Outro = queue[LENGTH(queue)-1]
    OUTPUT "ADA: " + str(Outro)
    WAIT(3)
    OUTPUT "ADA: Hopefully that helped clear things up! Have a nice day! Heading back to the menu now..."
    WAIT(2)
END SUBROUTINE
```

The last element is the outro, which is output to the screen

Ends the tutorial, but gives the user some information about what is happening

Railfence Encode/Decode

```
SUBROUTINE RF_EncodeDecode(encode=True)
    IF encode=True THEN
        OUTPUT "ADA: The Rail Fence cipher is complicated to encode, but it's simple with a couple of key pieces of information."
    ELSE
        OUTPUT "ADA: The Rail Fence cipher is complicated to decode, but it's simple with a couple of key pieces of information."
    END IF
    OUTPUT "ADA: One of these is the number of rails, the other is your text! "
    text = GetText()
    railCorrect = False
    WHILE railCorrect == False
      TRY
        IF encode=True THEN
            railNum = INPUT INTEGER "ADA: How many rails do you want to use to encode this with? \nYou: ")
        ELSE
            railNum = INPUT INTEGER "ADA: How many rails do you want to use to decode this with? \nYou: ")
        END IF
        rfRails = RailFence(ciphertext=text, railNo=railNum)
        guess = 1 + railNum
        rfPotentials = rfRails.GuessRails(guess)
        IF railNum IN rfPotentials THEN
            railCorrect = True
```

Depending on the value of encode, then different output to the screen (as a multi-use function)

Gets text using a standardised function

While the rail number isn't correct, then the code keeps repeating. If any errors come up across this entire section, then the loop starts again

This checks if the rail number is calculated to be okay to encode with

If it isn't correct, then the code rejects it and prints an error message

```
        ELSE
            IF encode=True THEN
                OUTPUT "ADA: In order to properly encode the Rail Fence cipher, you need a rail number that, when you do the modulus of the length of text by ((2*rails) -2), it should be 1. \nADA: Please try a different number, as this wasn't acceptable."
            ELSE
                OUTPUT "ADA: In order to properly decode the Rail Fence cipher, you need a rail number that, when you do the modulus of the length of text by ((2*rails) -2), it should be 1. \nADA: Please try a different number, as this wasn't acceptable."
            END IF
        END IF
    EXCEPT
        OUTPUT "ADA: That doesn't seem right. Could you try something different perhaps?"
END WHILE
IF encode=True THEN
    OUTPUT "ADA: Now I'm going to encode it for you."
    railEnc = RailFence(railNo=railNum, ciphertext=text)
    railEncoded = railEnc.encode()
    OUTPUT "ADA: Here is your encoded Rail Fence:", railEncoded
ELSE
    OUTPUT "ADA: Now I'm going to decode it for you."
    railDec = RailFence(railNo=railNum, ciphertext=text)
    railDecoded = railDec.decode()
    OUTPUT "ADA: Here is your decoded Rail Fence:", railDecoded
END IF
WAIT(3)
OUTPUT "ADA: Heading back to the Rail Fence menu now!"
WAIT(1.5)
END SUBROUTINE
```

Now all the information that has been gathered is correct and certified, then the code finally puts it into the class and encodes it

Returns to the menu

Compare

```
SUBROUTINE Compare()

    OUTPUT "ADA: Here, I'm going to show you some comparisons between the text you 'll enter and the values for a normal English piece of text."

    WAIT(1)

    OUTPUT "ADA: But, before I continue..."

    WAIT(0.5)

    Perctext = GetText()

    OUTPUT "ADA: I'm going to use the percentages for each letter in the text, as this is easier to visually compare."

    perc = Analysis(text=Perctext)

    percAnalysis = perc.GetCipherPercents()

    engPerc = perc.GetEnglishPercents()

    OUTPUT '''ADA: This'll be in the system: Letter | English | Ciphertext  ----------'''

    FOR letter IN engPerc

      OUTPUT "    " + letter + " | " + str(engPerc[letter]) + " | " + str(percAnalysis[letter])

    END FOR

    OUTPUT "ADA: Hopefully this was useful for you! Heading back to the Analysis Menu now..."

    WAIT(2)

END SUBROUTINE
```

Shows some rubric/text to tell you about the program

Gets text using standardised function

Using the Analysis class, retrieves the English values and the values for the text you have just entered

Prints the information out in a grid style

Returns you to the menu

PositiveResponse

```
  SUBROUTINE PositiveResponse(text):

    correctedtext = CorrectText(text)

    positives = ["YES", "Y", "OFCOURSE", "YUP", "YEAH", "YAH", "YH", "DONE"]

    IF correctedtext IN positives THEN

      RETURN True
```

Corrects the text, so it removes any extraneous information

There is a stored number of responses deemed positive

If the response was in the array, return the value True

whichCipher

```
SUBROUTINE whichCipher(text):

    caesaroptions = ["CAESAR", "CAESARCIPHER"]

    railfenceOptions = ["RAIL", "RF", "RAILFENCE"]

    substitutionOptions = ["SUBSTITUTION", "SUB", "ALPHABETICSUBSTITUTION", "AS"]

    transpositionOptions = ["TRANS", "COLUMNARTRANPOSITION", "TRANSPOSITION", "CT"]
```

Many options for different ways to say each cipher.

```
    analysisOptions = ["ANALYSIS", "ANALYSE", "CALCULATE", "FREQUENCYANALYSIS", "A",
"HELPANALYSE"]
    vigenereOptions = ["VIG", "VIGENERE", "VIGENERECIPHER", "V"]
    tutorialOptions = ["TUTORIAL", "HELP"]
    unknownOptions = ["UNKNOWN", "COLDCIPHER", "UNKNOWNCIPHER", "CRYPTOGRAPHY",
"DECRYPT"]
    quitOptions = ["DONE", "FINISHED", "STOP", "END", "EXIT", "QUIT"]
    correctedtext = CorrectText(text)
    IF correctedtext IN caesaroptions THEN
      RETURN "C"
    ELSE IF correctedtext IN substitutionOptions THEN
      RETURN "S"
    ELSE IF correctedtext IN railfenceOptions THEN
      RETURN "RF"
    ELSE IF correctedtext IN transpositionOptions THEN
      RETURN "TR"
    ELSE IF correctedtext IN analysisOptions THEN
      RETURN "A"
    ELSE IF correctedtext IN vigenereOptions THEN
      RETURN "V"
    ELSE IF correctedtext IN tutorialOptions THEN
      RETURN "TU"
    ELSE IF correctedtext IN quitOptions THEN
      RETURN "Q"
    ELSE IF correctedtext IN unknownOptions THEN
      RETURN "U"
    ELSE
      RETURN "ERROR"
    END IF
END SUBROUTINE
```

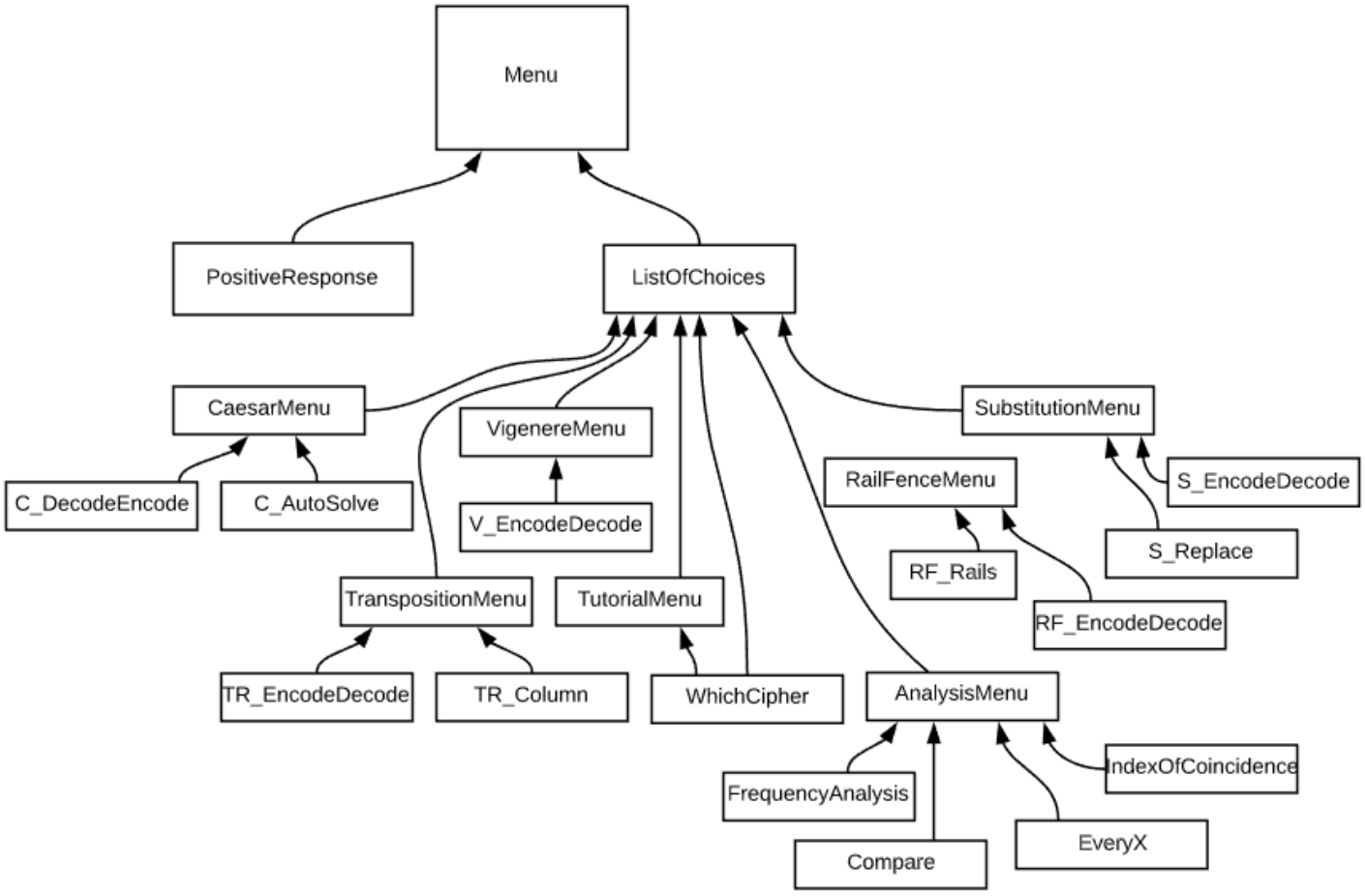
Corrects the text to remove extraneous information

If the response if found in any other the cipher lists, it returns the specific code for that cipher, for example, Transposition has the code 'TR'

If it isn't found in any of them, then the code returns an error message for the previous function to deal with

Hierarchy Chart

This is certainly the most complex of the Hierarchy diagrams. However, there are some small problems with it, as some specific granularity may be missing and need to be expanded on later, due to repetition of code. This will be expanded on in the Build, however complexity is not missing here because of it.



Class Definition

Class Name:	Menu	
Methods		
	Name	**Comments**
Public	ListOfChoices	The main looping function in the code, which provides access to the individual cipher menus (Caesar to Analysis)
	PositiveResponse	A function which returns a Boolean and compares against an array of strings deemed “positive”, returns True if it is.
	whichCipher	Using arrays of different cipher keywords, the function checks to see if the text is in any one of them. If so, returns the relevant keyword. If not, returns an error message.

	GetText	This can be used to just type in text, or load a text file instead of typing it. It checks thoroughly that the text input/chosen is correct. If the specific cipher requires it, remove all punctuation/erroneous characters.
	CorrectText	This removes all punctuation and strips the text to check that there is actually usuable text inside.
	ModeChoice	Using certain variables set to True, once the correct mode (from the input) has been identified, then checks to see if that mode is valid. Returns the specific code for the mode of choice e.g. “ENCODE” if an encoding option chosen
	TutorialMenu	This, once the correct cipher mode has been picked by WhichCipher, loops through the queue provided, displaying it in the pattern intro, decoding & explanation, then outro.
	CaesarMenu	The main Caesar menu, directs to the encode/decode function and the auto solve
	C_DecodeEncode	Depending on the Booelan value of ‘encode’, it prints different statements, but the working is the same until the input into the Class at the bottom.
	C_AutoSolve	This gets the text from the user, and prints all 26 options, to show them the potential permutations that it can be.
	TranspositionMenu	The main transposition menu, directs to encode/decode and the column guesser
	TR_EncodeDecode	This, depending on the value of ‘encode’, prints out different statements, but validation is mainly the same. Prints the encoded or decoded text to the screen.
	TR_Column	Based upon the length of the text, suggests the potential columns for the encoding or decoding.
	VigenereMenu	The main Vigenère menu, directs to the encode/decode function
	V_EncodeDecode	This, depending on the value of ‘encode’, prints out different statements, but validation is mainly the same. Prints the encoded or decoded text to the screen.
	SubstitutionMenu	The main substitution menu, directs to the encode/decode and the replacement suggester
	S_EncodeDecode	This, depending on the value of ‘encode’, prints out different statements, but validation is mainly the same. Prints the encoded or decoded text to the screen.

	S_Replace		This suggests which letters in the ciphertext to pair with the normal values of English. This uses percentages, meaning it may not be perfectly accurate
	RailFenceMenu		The main rail fence menu, which directs to the encode/decode function and the Rail Guesser
	RF_EncodeDecode		Depending on the Booelan value of 'encode', it prints different statements, but the working is the same until the input into the Class at the bottom.
	RF_Rails		Using the text input, calculates the rail numbers that are valid, given an input of the max number to check.
	AnalysisMenu		The main Analysis menu, directs to all the capabilities, including Frequency Analysis, IndexOfCoincidence, Compare and EveryX
	FrequencyAnalysis		Displays the frequencies of each letter from the text that you input
	IndexOfCoincidence		Displays the Index of Coincidence value for the text input.
	Compare		Displays the percentages of each letter of the ciphertext, next to the percentages you'd expect from English
	EveryX		Takes a number in, and then finds out the number of the occurrences
Attributes			
	Name	**Data Type**	**Comments**
Private/Public	**N/A**		**There are no class variables in this class**

Conversation with End User

Just after finishing my design, I decided to have another conversation with my end user, Dr Drape, to evaluate whether my proposed design still met his objectives. Here are some key points from this discussion:

What do you think about the current design?

This appears to be okay. It does look really complicated – I understand if this is necessary, but perhaps it could be simpler. You are doing a lot here, which can be positive and negative.

Is there anything key missing from the project?

No, the ciphers that you have included seem to be good. I would probably suggest you could explain things better in the tutorial – for example the Index of Coincidence is quite a hard concept to get your head around. It might be useful to explain it better, and perhaps explain the process of calculating it in the Tutorial.

An option that would be interesting would be that it analyses a piece of text and can tell user what cipher it potentially is. You might also want to allow a text file to input text from, as the ciphers are often quite long, so it would be a pain copying and pasting it in.

You might also want to signpost your menus better, because I can understand that a user might get lost in them. I wouldn't be averse to a simpler numbered menu system. However, the chatty nature may be nice for new users.

Do you have any concerns about the Build?

No, it seems to be doable within this timeframe, and it appears to fulfil all my criteria – from the design at least. It would be nice to see the full project working one it has been completed.

Build

```
import time
```

Cipher Class

```
class Cipher: ##Abstract Base Class, ABC
    def __init__(self,):
        self.plaintext = ""
        self.ciphertext = ""
        self.__alphabet = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n"
,"o","p","q","r","s","t","u","v","w","x","y","z"]

    def encode(self):
        pass
    def decode(self):
        pass

    @staticmethod
    def AsciiToLetter(ascval):
        return chr(ascval)

    @staticmethod
    def LetterToAscii(letter):
        return ord(letter)
```

Caesar Class

```
class Caesar(Cipher):
    def __init__(self, plaintext = "", ciphertext = "", shift = 0):
        self.__plaintext = plaintext
        self.__ciphertext = ciphertext
        self.__shift = shift #Initialises the variables for the Caesar class

    def encode(self):
        self.__ciphertext = self.solve(self.__plaintext) #Uses the function solve
to encode the caesar
        return self.__ciphertext

    def solve(self, text1): #Every letter that is in the alphabet is shifted, the
rest are left untouched
        text2 = ""
        for letter in text1:
            if letter.isalpha():
                text2 += self.shiftLetter(letter)
            else:
                text2 += letter
        return text2


    def shiftLetter(self, letter):
        value = self.LetterToAscii(letter) + self.__shift
        if letter.isupper():
            value -= self.LetterToAscii("A") #Takes away ascii value
            value = value % 26 # Makes sure it wraps around the alphabet
            value += self.LetterToAscii("A") # Adds back on ascii value
        elif letter.islower():
            value -
= self.LetterToAscii("a") ##Same but instead uses the lowercase ascii values
            value = value % 26
            value += self.LetterToAscii("a")

        newLetter = self.AsciiToLetter(value) #Converts the new value back into a
letter
        return newLetter

    def decode(self):
        self.__shift *= -
1 #Turns the shift negative, therefore it shifts the other way.
        self.__plaintext = self.solve(self.__ciphertext) #Puts the ciphertext into
 solve
        return self.__plaintext

    def autoSolve(self):
        texts = []
        for x in range(0,26):
            self.__shift = x
            self.decode()
            texts.append(self.__plaintext)
        return texts
```

Substitution Class

```
class Substitution(Cipher):
    ## Use translate and make trans -
> easy way to substitute the alphabet once you have it correct
    #e.g hello = string.maketrans(start,end)
    #  string.translate(hello)
    #OR -> string.translate(string.maketrans(start,end))

    def __init__(self, ciphertext = "", cipherbet = "", plaintext = ""):
        self.__plaintext = plaintext #Initialises the variables for the Substituti
on class
        self.__ciphertext = ciphertext
        self.__plainbet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" #
In order to make sure the alphabet works with the maketrans, needs both upper and
lower case.
        self.__cipherbet = cipherbet


    def encode(self):
        self.__ciphertext = self.__plaintext.translate(self.__plaintext.maketrans(
self.__plainbet,self.__cipherbet))
        #Using the inbuilt maketrans and translate functions, the first is substit
utued for the second
        # (so plainbet 'a' is substituted for whatever the first letter in the cip
herbet is)
        return self.__ciphertext

    def decode(self):
        self.__plaintext = self.__ciphertext.translate(self.__ciphertext.maketrans
(self.__cipherbet,self.__plainbet))
        # The first letter in the cipherbet is substituted for 'a' from the plainb
et
        return self.__plaintext

    def SuggestReplacements(self):
        self.__analyse = Analysis(text = self.__ciphertext) #Creates an Analysis o
bject
        #self.__cipherFreq = self.__analyse.GetCipherFrequencies
        #self.__cipherPerc = self.__analyse.GetCipherPercents
        suggestedList = self.__analyse.PairUpValues() #Uses a function in the anal
ysis class to pair up the values in the text.
        return suggestedList
```

Vigenere Class

```
class Vigenere(Cipher):

    def __init__(self, plaintext = "", ciphertext = "", key = ""):
        self.__ciphertext = ciphertext
        self.__plaintext = plaintext #Initialises thhe variables for the Vigenere
Class
        self.__key = key

    def encode(self):
        self.__ciphertext = "" # Resets the ciphertext
        x = 0
        for letter in self.__plaintext:
            if letter.isalpha(): #If it's a letter
                newLetter = self.shiftLetterUp(letter, self.__key[x]) #Shifts the
letter
                try:
                    self.__key[x+1] #cycles through the key, checking if it's gone
 over the edge of it
                    x += 1
                except:
                    x = 0 #goes back to the start of the key
                self.__ciphertext = self.__ciphertext + newLetter #Adds it onto th
e text
            else:
                self.__ciphertext = self.__ciphertext + letter
        return self.__ciphertext

    def shiftLetterUp(self, letter, keyChar):
        if letter.islower():
            #Adds on the key value to the letter, then loops around the alphabet,
and turns it back into a letter
            shifted = self.AsciiToLetter((((self.LetterToAscii(letter)-
97) + (self.LetterToAscii(keyChar.lower())-97)) %26) + 97)
        elif letter.isupper(): # Same but using the uppercase values
            shifted = self.AsciiToLetter((((self.LetterToAscii(letter)-
65) + (self.LetterToAscii(keyChar.upper())-65)) %26) + 65)
        return shifted

    def shiftLetterDown(self, letter, keyChar):
        if letter.islower(): #Similar to ShiftLetterUp but instead takes away the
key value
            shifted = self.AsciiToLetter((((self.LetterToAscii(letter)-
97) - (self.LetterToAscii(keyChar.lower())-97)) %26) + 97)
        elif letter.isupper():
            shifted = self.AsciiToLetter((((self.LetterToAscii(letter)-
65) - (self.LetterToAscii(keyChar.upper())-65)) %26) + 65)
        return shifted

    def decode(self):
        self.__plaintext = "" # Resets the plaintext
        x = 0
        for letter in self.__ciphertext: #Instance of using a STRING like an ARRAY
            if letter.isalpha(): #If it's a letter
                newLetter = self.shiftLetterDown(letter, self.__key[x])
                try:
                    self.__key[x+1] #Cycles through the key, checking if you've go
ne over
                    x += 1
```

```
                except:
                    x = 0
                self.__plaintext = self.__plaintext + newLetter
            else: #If it's not a letter ignore and add to text
                self.__plaintext = self.__plaintext + letter
        return self.__plaintext

    @staticmethod
    def ClosestToEnglish(listOfOptions):
        BestGuess = 10000 ##A large number to start off with, as if 0, then it'll
never change
        BestText = ""
        englishValue = 1.73 #The value of a normal english text on average
        for option in listOfOptions:
            analyse = Analysis(text= option) #Creates an Analysis Object
            ioc = analyse.GetIoCValue() #Calculate IOC values
            ###DUE TO IOC IT WORKS BEST ON LONG TEXTS###
            close = englishValue - ioc
            if close < 0:
                close = close*-1 ###want a scalar value, no plus/minus
            if BestGuess > close: # If close is smaller, then it must be "better"/
more like English
                BestGuess = close
                BestText = option #set this closest text as the best text
        return BestText
```

Transposition Class

```
class Transposition(Cipher):

    def __init__(self, ciphertext="", plaintext="", columns=0, order = [], reverse
Order = []):
        self.__ciphertext = ciphertext
        self.__plaintext = plaintext #Initialises variables for the Transposition
class
        self.__columns = columns
        self.__columnOrder = order #In style [0,1,2,3,4,5]
        if columns != 0: #Columns may not have been set
            if reverseOrder == []:
                self.reversePermutation() #If it hasn't been set already, set the
reverse order
            else:
                self.__reverseOrder = reverseOrder

    def encode(self):
        words = ""
        for index in self.__columnOrder:
            #Uses the normal order
            for x in range(0, len(self.__plaintext), self.__columns):
                #from the beginning of the plaintext, skipping by the columns
                try:
                    words += self.__plaintext[x+index]
                    #adds the letter onto the text tally
                except:
                    break
        return words

    def decode(self):
        words=""
        #Calculates the number of rows in the text
        rows = len(self.__ciphertext)//(self.__columns)
        #print(self.__columns)

        for x in range(0, rows):
            #for every row
            for index in self.__reverseOrder:
                #pick from the index of your reverse combo
                try:
                    #you want to skip by index*rows each time
                    words += self.__ciphertext[index*rows + x]
                except:
                    #print(x)
                    pass
        return words

    def FindPossibleColumns(self):
        factors = []
        for i in range(1, (len(self.__ciphertext)) + 1):
            #finding the factors in the length of the text
            if (len(self.__ciphertext)) % i == 0:
                factors.append(i)
        return factors

    def reversePermutation(self):
        index = []
```

```
        for item in self.__columnOrder: #Creates a list of the column order and it
's index (TUPLES)
            position = self.__columnOrder.index(item)
            index.append((item,position)) #Instance of a TUPLE
        reverse = []
        for i in index: #Creates a list filled with 0s, in order to avoid errors
            reverse.append(0)
        for elem in index: #The number of the order now becames the position
            reverse[elem[0]] = elem[1]
        #print(reverse)
        self.__reverseOrder = reverse #Sets as the variable
        return self.__reverseOrder

    def takeSecond(self, elem):
        return elem[1] #returns the second element
```

RailFence Class

```
class RailFence(Cipher):
    ## characters = (c+1) + (r-2)(2c) + c
    # c = cycles, r = rails
    def __init__(self, ciphertext = "", plaintext= "",railNo = 0):
        self.__ciphertext = ciphertext
        self.__plaintext = plaintext #Initialises the variable for use in the rail
fence class
        self.__railNo = railNo
        self.__railorder = []
        for x in range(0, railNo): #Creates an array with the values of the railfe
nce (cycles)
            self.__railorder.append(x) #Will look like [0,1,2,3,2,1]
        for x in range(railNo-2, 0,-1): #Goes back up the rails
            self.__railorder.append(x)
        #print(self.__railorder)
        #the characters -1 divided by 2r-1 = cycles
        # OR characters divided by len(railorder)
        if self.__ciphertext == "":
            self.__cycles = len(self.__plaintext)/len(self.__railorder)
        else:
            self.__cycles = len(self.__ciphertext)/len(self.__railorder)
        self.__maxCycle = (2*self.__railNo) -2 #Max number of letters in a 'cycle'

        pass

    def encode(self):
        currentrail = 0
        self.__ciphertext = "" #resets ciphertext
        self.FirstRail() #encodes the first rail (special case)
        currentrail+=1 #moves onto the next rail
        while currentrail != (self.__railNo-1): #While it's not the last rail
            self.MiddleRail(currentrail, self.__maxCycle-
currentrail) #Encodes the middle rail (general case)
            #puts into middlerail the two opposite numbers (so 2,4 if rail=5)
            currentrail+=1 #moves onto the next rail
        self.FinalRail() #encodes the last rail (special case)
        #print(self.__ciphertext)
        return self.__ciphertext

    def FinalRail(self):
        words = ""
        for x in range(0,len(self.__plaintext), int(self.__maxCycle)): #loops thro
ugh the plaintext, skipping by the maxcycle
            try:
                words += self.__plaintext[x + (self.__railNo-
1)] #Takes every value but increased by railno -1
            except:
                break
        #print(words)
        self.__ciphertext += words

    def MiddleRail(self, first, second):
        words = ""

        for x in range(0,len(self.__plaintext), int(self.__maxCycle)): #loops thro
ugh the plaintext skipping by maxCycle
            try:
                words += self.__plaintext[x+first] #The first (down) letter
```

```
                words += self.__plaintext[x+ second] # the second (up) letter
            except:
                break
        #print(words)
        self.__ciphertext += words

    def FirstRail(self):
        words = ""
        for x in range(0,int(self.__cycles)+1): #Loops through values for total nu
mber of cycles plus one
            try:
                words += self.__plaintext[x * (((2*self.__railNo)-1)-
1)] #Using algebra previously worked out, picks the next top line letter each time
            except:
                break
        #print(words)
        self.__ciphertext += words

    def decode(self):
        rails = self.SplitIntoRows() #Chunks it into rails

        #Take one from each going down then back up
        text = self.PickingValues(rails)

        #returns and adds the plaintext as the ciphertext.
        self.__plaintext = text
        return self.__plaintext

    def SplitIntoRows(self):
        toprail = ""
        bottomrail = "" #Initialises Varaible
        rails = [] #an ARRAY to store the rails.
        counter = -1 #set to -
1 as we add to counter inside loop and need 0 to begin with
        n = (len(self.__ciphertext)-1)//((2*self.__railNo)-
2) #n is number of characters per row (used as part of the algebra)
        #print(n)
        #chunk into rails, bottom is n, top is n+1, middles are 2n
        try:
            for x in range(0, n+1):
                counter +=1
                toprail += self.__ciphertext[counter] #the top rail is n+1 charact
ers
            rails.append(toprail)
            for i in range(0, self.__railNo-2):
                temp = ""
                for x in range(0, n *2 ): #middle rail is2n characters as up & dow
n
                    counter +=1
                    temp+= self.__ciphertext[counter]
                rails.append(temp)
            for x in range(0, n): #bottom rail is only n characters
                counter +=1
                bottomrail += self.__ciphertext[counter]
            rails.append(bottomrail)
        except:
            pass
        #print(rails)
        return rails
```

```
    def PickingValues(self, rails):
        current = 0
        up = True
        text = ""
        while rails[0] != "": #whilst the toprail is not empty
            listOfChars = rails[current] #take the currrent row
            #print("Left:", listOfChars)
            first = listOfChars[0] # Using a functional programming "head" style f
unction
            fixed = listOfChars[1:] # Using a functional programming "tails" style
 function
            rails[current] = fixed #returns the list back into the array
            #print(current)
            #print(first)
            text+= first #adds onto the running text tally
            if up == True: #takes the next rail, depending on if going up or down
                current+=1
            else:
                current-=1
            if current>= len(rails): #if it's larger than the rail length (gone ov
er the bottom)
                current -=2 #goes back up to the right row
                up = False #Realises it's now reached the bottom so time to go up
            elif current <0: #If it's now gone over the top,
                current+=2 #goes back down to the right row
                up = True #Realises that its time to start going down as it's reac
hed the top
        #print("finished")
        #print
        return text

    def GuessRails(self, topGuess):
        potentials = []
        for x in range(2,1+topGuess): #between the two values, 2 and your top gues
s, check if the text can be a complete railfence
            #print(x, len(self.__ciphertext) % ( (2*x) -2))
            if (len(self.__ciphertext) % ( (2*x) -
2)) == 1: #the modulus of the length of text by 2*rails -
2 should be one if complete cycle
                potentials.append(x) #adds it onto a running rail length selection
        #print(potentials)
        return potentials
```

Tutorial Class

```
class Tutorial:
    def __init__(self):
        ##Caesar Section
        self.__CaesarPlaintext = "HARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY W
EARING SUNGLASSES"
        self.__CaesarCiphertext = "XQHHO XQI JXU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO
MUQHYDW IKDWBQIIUI"
        self.__CaesarShift = 16
        self.__CaesarQueue = ["You start by taking the first letter of the ciphert
ext. This then needs to be shifted back down the alphabet by the shift you used or
iginally.",["So, we move the “X” back to an “H”.","HQHHO XQI JXU SQIU VYBU. CUUJ Q
J JMUBLU JXYHJO MUQHYDW IKDWBQIIUI"], ["Once we’ve figured out that letter, we mov
e onto the next. In this example, it’s the “Q”. This gets shifted back to an “A”.
","HAHHO XQI JXU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI"], ["The rest
 of this cipher is rather the same, just shifting a letter back through the text.
Here is the mapping of the characters, so it’s easier to see: ","ABCDEFGHIJKLMNOPQ
RSTUVWXYZ (English) \nQRSTUVWXYZABCDEFGHIJKLMNOP (Ciphertext) "],["I’ll show you t
he rest of the cipher decryption now. ","HARHHO XQI JXU SQIU VYBU. CUUJ QJ JMUBLU
JXYHJO MUQHYDW IKDWBQIIUI \nHARRO XQI JXU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW
 IKDWBQIIUI \nHARRY XQI JXU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \n
HARRY HQI JXU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAI JXU
SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS JXU SQIU VYBU. CUU
J QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS TXU SQIU VYBU. CUUJ QJ JMUBLU JX
YHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THU SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW I
KDWBQIIUI \nHARRY HAS THE SQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHA
RRY HAS THE CQIU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CA
IU VYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASU VYBU. CUUJ
QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE VYBU. CUUJ QJ JMUBLU JXYH
JO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FYBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKD
WBQIIUI \nHARRY HAS THE CASE FIBU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARR
Y HAS THE CASE FILU. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE
 FILE. CUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI "],["Almost there… ","HARRY HAS TH
E CASE FILE. MUUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. M
EUJ QJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEEJ QJ JMUBLU
JXYHJO MUQHYDW IKDWBQIIUI  \nHARRY HAS THE CASE FILE. MEET QJ JMUBLU JXYHJO MUQHYD
W IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AJ JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \
nHARRY HAS THE CASE FILE. MEET AT JMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE
 CASE FILE. MEET AT TMUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. ME
ET AT TWUBLU JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWEBLU J
XYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELLU JXYHJO MUQHYDW
IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVU JXYHJO MUQHYDW IKDWBQIIUI \nH
ARRY HAS THE CASE FILE. MEET AT TWELVE JXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE C
ASE FILE. MEET AT TWELVE TXYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET
 AT TWELVE THYHJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THI
HJO MUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRJO MUQHYDW IK
DWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTO MUQHYDW IKDWBQIIUI \nHAR
RY HAS THE CASE FILE. MEET AT TWELVE THIRTY MUQHYDW IKDWBQIIUI \nHARRY HAS THE CAS
E FILE. MEET AT TWELVE THIRTY WUQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET A
T TWELVE THIRTY WEQHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRT
Y WEAHYDW IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARYDW IKDW
BQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARIDW IKDWBQIIUI \nHARRY
 HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARINW IKDWBQIIUI \nHARRY HAS THE CASE
FILE. MEET AT TWELVE THIRTY WEARING IKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT
TWELVE THIRTY WEARING SKDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY
WEARING SUDWBQIIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARING SUNWBQ
IIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARING SUNGBQIIUI \nHARRY H
AS THE CASE FILE. MEET AT TWELVE THIRTY WEARING SUNGLQIIUI \nHARRY HAS THE CASE FI
LE. MEET AT TWELVE THIRTY WEARING SUNGLAIIUI \nHARRY HAS THE CASE FILE. MEET AT TW
```

```
ELVE THIRTY WEARING SUNGLASIUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WE
ARING SUNGLASSUI \nHARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY WEARING SUNGLASS
EI "],["Ta-
da!! Our decoded Caesar cipher. ","HARRY HAS THE CASE FILE. MEET AT TWELVE THIRTY
WEARING SUNGLASSES"], "This cipher is the easiest cipher you'll probably encounter
, as it's quite easy to decode. For this example, we know the shift, which is 16.
If we didn't know this, we could try 1 to 26 systematically, as this cipher is cyc
lical, meaning it has a finite number of options."]

        ##Vigenere Section
        self.__VigNoKeyQueue = ["Today we'll be looking at how to decode a Vigener
e Cipher if you do not know the key, but instead a small part of the text. We know
 "HARRY" is the first in the word in the text, but not what the key is, or how lon
g it is. For context, this is a message from two spies in the world of George Orwe
ll's 1984, so we may be able to guess some key words.",["The way you decode withou
t a key, is by putting the crib (what you call a piece of known text) in various p
ositions through the text. Because we know that it is at the start of the text, we
 don't need to keep shifting the crib around, as it's the first word. We'll start
with just the crib like normal and see how that works.","OBRIE, HWQ BBLGEOK JW SXW
MPVJW. AF KB VYPUEJ. SSQ"], ["That didn't produce anything recognisable, however "
HARRY" went to "OBRIE" Let's try with the crib as "BIGBROTHER" (and without HARRY
as we know what that is) to see if that's in there.","NOB RIENOBR PO DNDFWFWD. GX
VR CRWERQ. YKB"],["Can you see that repeating pattern? "NOBRIENOBR", and when comb
ined with the part from before, we can see that the start of the text is "OBRIE, N
OB RIENOBR". From this we can suggest the key is OBRIEN, which makes sense as he's
 an influential character from the book. Next,  try your guessed key on the text t
o see the result.","HARRY, BIG BROTHER IS WATCHING. GO TO GROUND. MEG"], "And perf
ect! You've decoded this without using a key, and now you've cracked the code. The
 Vigenère is a lot harder than the Caesar cipher, however if you know a small sect
ion of the text, you can potentially find the key, then crack the whole thing. Thi
s may feel like cheating, but this is a valid way of decoding real world encryptio
ns. In World War 2, the code-
breakers at Bletchley Park used a crib when deciphering the Enigma codes, because
the Germans started every morning's cipher with "The weather is…", so they knew wh
at the code would start with, and therefore easier to crack!"]
        self.__VigNoKeyPlaintext = "HARRY, BIG BROTHER IS WATCHING. GO TO GROUND.
MEG"
        self.__VigNoKeyCiphertext = "VBIZC, OWH SZSGVFI QW JOUTPMAU. HF BS TFPLVH.
 ZSH"
        #self.__VigNoKeyCrib = "OBRIEN"
        self.__VigNoKeyKey = "OBRIEN"

        ##Rail Fence Section
        self.__RailFenceQueue = ["This cipher is one of the most difficult to solv
e, as it looks on paper to be a transposition, however the way you unravel it is v
astly different.  ",["Due to the length of the text, we can guess that this is a 4
-
rail Railfence, as to calculate the probable rails, you want a modulus of (2*rails
 -2) that equals 1, as the number of letters in the cycle = ((2*rails -
2) * cycles + 1).   ","1 = 25 MOD (2*4 -2). Therefore it must be a 4-
rail.  "], ["In order to decode a railfence, the main thing is that you have to ke
ep track of which rail you're on, and then whether you are going up or down them.
That is our first point of call, break the text into 4 different "rails": first (t
op) has n+1 characters, the middle lines have 2n characters, and the last (bottom)
 has n characters. This only works is the Railfence has a complete cycle, meaning
it goes all the way down, all the way up, and ends on the top rail. ","'HUAEG', 'E
TCLIEME', 'LSKNRREM', 'PIF!'"], ["Next we begin by taking values down the rows, an
d then back up them, by keeping track of whether we have reached the end of a cycl
e. Remember, a cycle goes through the top line, once, the middle lines twice, and
```

```
the bottom once. I’ll show you the first cycle now: ","HELPST "],["Once you’ve got
 all your cycles you then concatenate all your cycles together (plus one extra cha
racter for the last element of the top row). ","HELPST, UCKINL, AIRFRE, ME!ME, G "
],["And here we are – the final result. ","HELPSTUCKINLAIRFREEME!MEG "], "A Rail f
ence cipher may look complicated to decode, but as soon as you know what you need
to do to decode it, it’s as easy as a simple transposition cipher."]
        self.__RailFencePlaintext = "HELPSTUCKINLAIRFREEME!MEG"
        self.__RailFenceCiphertext = "HUAEGETCLIEMELSKNRREMPIF!"

        #Unknown Section
        self.__UnknownCipherQueue = ["For this tutorial, we know somethings and no
t others. We have no idea which cipher this is, and therefore no idea on how to de
code – yet. By looking at the known parts of the text, we can make inferences and
find potential ciphers. The three categories that we will deal with are: Monoalpha
betic, Polyalphabetic and Transposition. The first of these, Monoalphabetic, will
be very obvious, as it has a frequency analysis and IOC similar to English, but th
e percentages of letter are in the wrong place. The second, Polyalphabetic, will h
ave a “flat” frequency analysis (all values same or similar), and a completely dif
ferent IOC value to English. The last, Transposition, will have a similar IOC to E
nglish, and the percentages will be the same, but the text will look nothing like
English.",["We are going to start by looking at the frequency analysis and the Ind
ex of Coincidence. You can see that we have an IOC remarkably similar to the base
English value (1.75) which suggests it’s not a Polyalphabetic cipher. Our Frequenc
y Analysis has shown that ‘e’ is the most common letter, closely followed by ‘t’.
This suggests it is not a monoalphabetic cipher either, as the letter frequencies
are approximately on the right letters.  ","Percentages: {'a': 6.1, 'b': 2.07, 'c'
: 2.397, 'd': 4.575, 'e': 13.181, 'f': 2.07, 'g': 3.05, 'h': 7.625, 'i': 7.19, 'j'
: 0.109, 'k': 0.109, 'l': 3.595, 'm': 1.961, 'n': 6.1, 'o': 8.279, 'p': 2.07, 'q':
 0.0, 'r': 6.645, 's': 3.922, 't': 10.458, 'u': 2.614, 'v': 1.089, 'w': 3.813, 'x'
: 0.0, 'y': 0.98, 'z': 0.0} \n IOC: 1.75 (3 s.f.)"], ["Now we know that it’s a Tra
nsposition, we can guess what type of Transposition it is. Normally if you’re work
ing with the Cipher Challenge, there are very few Rail Fence ciphers so let’s assu
me it’s a normal (Columnar) Transposition until we find our otherwise. Let’s take
a look at the length of the text next. This can help tell you what columns you’ll
be using, as often they fill a complete grid, so their column number is a factor o
f the length.","Length: 1,160 \nFactors: [1, 2, 4, 5, 8, 10, 20, 29, 40, 58, 116,
145, 232, 290, 580, 1160]"],["Wow, that’s a long list.  Now we can look at the tex
t to see if there are any clues to which of these to pick… Do you see that repeati
ng section of Uppercase letters? This happens five times, at various points in the
 text. Let’s attempt using 5 columns now, and see if that observation has helped u
s.","pailo  shern eti WTGTDWBRRNH HOIIO  BEedriigfaHuole ionIsus wn etarategsn ic
 ihabomtwedtoho s dhtilt itsorc nht eWe eNH Hoe enrrg  ie hetwti hrdo itd eeeuPelt
  s ame dlvmevfhese rtsirtciat t.ur  e.urwot  dcafvYid elow,n stnreeruo .Hedduuotm
 rigl  t-
NH HOIIO  BEWTGTDWBRRr  nl  geltplanfit ritrgt irds  e  iatoneruro am eueip aoees
thHd ow,ahea ushrw  BErthfeoi mnfnWe  ihaoe itot,edr. go d jtaHdmdw lemde avenp h
saihonlhis gitcd giathtc oloeogosslrh  y, e ryen y hlouyr tppingapsOIIO  BEWTGTDW
BRRNH Hvneafnlp oneegw a as e ifsrlowornuaetafnti f n te  tpdedntnrae.do heeektisl
 hetWTGT hrrifwn,edr.thn  d,we nowiaofcho cuemt .hot-
liaotei n poe etc  a osifoc,yltoc n gtl e e m ecuf eerruootheott.in  psvhop,n anca
   BEWTGTDWBRRNH HOIIO  oa,lhae d  i g c adnhi hpc s mdrttnl pg yt mhsptat laabn r
eoee d,e uewtwsse rDWBRR herdmtiaofchrwoterrthd nh  ie Thlw huhme i os  i,nhdet a-
eelmaneleneThmha Thms ihobnerruhdusy iefe srl  dgos svtleeoa tirealDWBRRNH HOIIO
  BEWTGToavg ia .c hfntep.wb,cetooauw noaohhi oe d, aee tor segna ep gr no vbs  a
e.thoOIIO,we a  itd eee enh y he g  mnfnThtiogisee ct-
uthct  e r tp- n ettd r lhteelihte anauecd . tgcf alvoabo atwb eu "],["So, recap.
We know it’s a Transposition, it’s got 5 columns, now we need to figure out the or
der of those columns (if they’re even jumbled up). Here’s the text, once it’s been
 put through without changing the column order.  ","pH isaehn idl sldo vouupt uysl
```

orvest hehmtooe pparrp, niint gn ielgart anei pca tsalW-
O DTNI WGHI BT OBRDH ERWO WNBI THRIBG ROETHN WDOH TWI GBIHBTROOEDR IWWN ITBH OGR
B TRHE DNOW WHITBB IGERHOTeRv odrn ar eovi aaginf, glnlif lhaa pa Hg e.ueo colnd l
te hepe f lginiaw ton genfa pIi c.sta wu sabsr d, iencwt henriit gf oetshot rpaail
curro wadws tso ne rmog ndaseuron ath ethiitnicaal tf oionpehntg aei dbr y,ouft
mr atonmew heeats dmeptt tooe arhutt oep s idlespeag dandanbahotn ten eierrpls
ae tteog h.eriHde tdo ns doooh, rweevc,e b aeusnhke hetw tait swaeulseWs s.ehhe
t re hewtroN WDOH TWI GBIHBTROoE R,erh w trheehre nfirarefd rowm ginti ,it meadi
ndo efrfe n.cehWtheeehr t nwew onti thihde a,ryhowr reethd heoind to giown tti
h d,a m eo nedfifercene.h Tu oThPg hteoclil uwotde g mhi jtusst he a.meaHhe mdo c
emtit d- -dwloul istvla hmeo cemtitvde, fein h heeand sv ereept no trpeapt --
shee istenracl ti mech atioantan edtlol hsertiin .sfelu oThrgcht i,me tyheeclal.d
t iu oThrgchtwi meoans tt a hgin tthadclouc beaoencfl edvoer Ye r.iomu dg ht oed
gescuclsusfolfy wr a,heiln eev rfosyreat,u bn osoreor e tlaerh trye wueo bontd
tge.y.ou "],["Hmm, looks a lot like gobble-de-
gook. Let’s try a different one, but before we do that. Take a closer look at the
start of the half-
decoded text. Do you see “pH isaehn idl sldo vouupt”? It seems like ‘H’ is meant t
o be the start of the word, let’s move things around then to get that. ","H ispehn
adl sido vluuptouysl orve t hesmtooh ppaerp, riintngn i lgare anet pcaitsal -
O DWNI WTHI BG OBRTH ERDO WNWI THBIBG ROETHR WDON TWIH GBI BTROHEDR OWWN ITBH IGR
BOTRHE DNOW WHIT B IGBRHOTERv oern ad eovr aaginf, ilnlig lhaf pa ag e.Heo culnd o
te hlpe felgin aw tin geofa pni c.Ita ws sabur d,sienc t hewriitngf o tshoe rpatil
cuaro wrdws aso nt rmoe ndageuros athn eth itniiaal ctf o onpeintg hei dar y,buft
or amonmet hewats emeptd tote aroutt hep soidle peags dan anbadotn hen eterrpis
ae lteogth.er Hde ido nt dosoh, oweevr,e bcaeus hke netw hait t swa ulsees s.Whhe
tere h wtroe WDON TWIH GBI BTROHE R,orh wetrhe hre efiranefd rowm rintig ,it mead
ndo ifrfeen.ce Wthehehr e nwet onwi ththde ia,ry owr heethr hedind oto g own iti
h t,a mdeo n dfifercene.h Te oThug htPoclie uwolde gt mhi jtus t hesa.me Hhe ado c
mmtited- - wloud istlla hveo cmmtitede, vein f hehand ev ersept eno t peapr --
thee ssteniacl ri meth atcoantin edalol thser iin tsfel. oThugchtri,me tyhe claled
t i. oThugchtri mewans ot athgin ttha cloud becoencal edfoer ve r.Yomu ig htdoed
g scucesusfllfy or awheil, eevn rfo yreas,u bt osoneor r tlaerh teye wreo buntd o
tge y.ou. "],["Now that’s starting to look like English. You can see “y.ou.” at t
he bottom, and “scucesusfllfy”; you and successfully respectively. Let’s twiddle a
little more and see what it ends up like. I’ll try a column order of 1, 3, 4, 2,
0 next. ","Hi spenh ad lsid ovluputousyl ovre th esmotoh ppaer,p rinitng ni lagre
naet cpaitasl - ODWN IWTH IBG BORTHE RDOW NWIT HBIGB ROTEHR DWON WTIH BGI BRTOHERD
OWNW ITHB IG RBOTHRE DONW WIHT BI GBROHTER voer nad oevr aagin,f illnig hlaf ap a
ge .He oculdn ot ehlp efelign a wting eof apnic .It aws asburd ,sinec th ewriitng
fo thsoe praticluar owrdsw as ont mroe dnageruos tahn teh intiiala ct fo opneingt
he idary ,butf or amomnet h ewast empetd t otea routt he psoilde paegs adn abnado
nt he neterrpisea ltoegthe.r Hed id ont d oso,h oweevr, ebcaues hek newt hati t ws
a usleess .Whehter eh wrtoe DWON WTIH BGI BRTOHER ,or hwethre her efrianedf romw r
itnig i,t maed nod iffreenc.e Whteherh e wnet o nwit hthed iar,y orw hetehr h edid
n ot og onw ithi t, amde on differecne. hTe Tohugh tPolcie wuold egt hmi juts th e
sam.e Heh ad ocmmitted -
- wolud sitll ahve ocmmitted,e veni f h ehadn eve rsetp en ot paepr - -
thee ssetnialc rim etha tconatine dallo thesr ini tsefl. Tohughctrim,e thye called
ti. Tohughctrim ewasn ot athign thta colud b econecale dfore ver .Youm igh tdode
g successufllyf or awhiel, eevn fro yeras, ubt soonero r ltaer htey ewre obundt o
gte yo.u. "],["Yes! This is the right order to decrypt it. Here’s your completed
text: ","His pen had slid voluptuously over the smooth paper, printing in large ne
at capitals - DOWN WITH BIG BROTHER DOWN WITH BIG BROTHER DOWN WITH BIG BROTHER DO
WN WITH BIG BROTHER DOWN WITH BIG BROTHER over and over again, filling half a page
. He could not help feeling a twinge of panic. It was absurd, since the writing of
those particular words was not more dangerous than the initial act of opening the
diary, but for a moment he was tempted to tear out the spoiled pages and abandon
the enterprise altogether. He did not do so, however, because he knew that it was

```
useless. Whether he wrote DOWN WITH BIG BROTHER, or whether he refrained from writ
ing it, made no difference. Whether he went on with the diary, or whether he did n
ot go on with it, made no difference. The Thought Police would get him just the sa
me. He had committed -
- would still have committed, even if he had never set pen to paper -
- the essential crime that contained all others in itself. Thoughtcrime, they call
ed it. Thoughtcrime was not a thing that could be concealed for ever. You might do
dge successfully for a while, even for years, but sooner or later they were bound
to get you.. "], "Congratulations, you've cracked a cipher from start to finish! T
his is often a challenege for even the best code-
breakers, as you have to make so many logical leaps and guesses. Keep on breaking!
 "]
        self.__UnknownCipherCipherText = "pailo  shern eti WTGTDWBRRNH HOIIO   BEe
driigfaHuole ionIsus wn etarategsn ic ihabomtwedtoho s dhtilt itsorc nht eWe eNH H
oe enrrg  ie hetwti hrdo itd eeeuPelt  s ame dlvmevfhese rtsirtciat t.ur  e.urwot
 dcafvYid elow,n stnreeruo .Hedduuotm rigl  t-
NH HOIIO   BEWTGTDWBRRr  nl  geltplanfit ritrgt irds  e  iatoneruro am eueip aoees
thHd ow,ahea ushrw   BErthfeoi mnfnWe  ihaoe itot,edr. go d jtaHdmdw lemde avenp h
saihonlhis gitcd giathtc oloeogosslrh  y, e ryen y hlouyr tppingapsOIIO   BEWTGTDW
BRRNH Hvneafnlp oneegw a as e ifsrlowornuaetafnti f n te  tpdedntnrae.do heeektisl
 hetWTGT hrrifwn,edr.thn  d,we nowiaofcho cuemt .hot-
liaotei n poe etc  a osifoc,yltoc n gtl e e m ecuf eerruootheott.in  psvhop,n anca
   BEWTGTDWBRRNH HOIIO  oa,lhae d  i g c adnhi hpc s mdrttnl pg yt mhsptat laabn r
eoee d,e uewtwsse rDWBRR herdmtiaofchrwoterrthd nh  ie Thlw huhme i os  i,nhdet a-
eelmaneleneThmha Thms ihobnerruhdusy iefe srl   dgos svtleeoa tirealDWBRRNH HOIIO
  BEWTGToavg ia .c hfntep.wb,cetooauw noaohhi oe d, aee tor segna ep gr no vbs   a
e.thoOIIO,we a  itd eee enh y he g  mnfnThtiogisee ct-
uthct  e r tp- n ettd r lhteelihte anauecd . tgcf alvoabo atwb eu"
        self.__UnknownCipherPlainText = "His pen had slid voluptuously over the sm
ooth paper, printing in large neat capitals - DOWN WITH BIG BROTHER DOWN WITH BIG
BROTHER DOWN WITH BIG BROTHER DOWN WITH BIG BROTHER DOWN WITH BIG BROTHER over and
 over again, filling half a page. He could not help feeling a twinge of panic. It
was absurd, since the writing of those particular words was not more dangerous tha
n the initial act of opening the diary, but for a moment he was tempted to tear ou
t the spoiled pages and abandon the enterprise altogether. He did not do so, howev
er, because he knew that it was useless. Whether he wrote DOWN WITH BIG BROTHER, o
r whether he refrained from writing it, made no difference. Whether he went on wit
h the diary, or whether he did not go on with it, made no difference. The Thought
Police would get him just the same. He had committed -
- would still have committed, even if he had never set pen to paper -
- the essential crime that contained all others in itself. Thoughtcrime, they call
ed it. Thoughtcrime was not a thing that could be concealed for ever. You might do
dge successfully for a while, even for years, but sooner or later they were bound
to get you.."
        self.__UnknownCipherColumns = 5
        self.__UnknownCipherColumnOrder =  [4,0,3,1,2]
        self.__UnknownCipherReverseOrder =  [1, 3, 4, 2, 0]
        pass

    ##Order: Queue, Plaintext, Ciphertext, (extra stuff)
    def getCaesarInfo(self):
        return self.__CaesarQueue,  self.__CaesarPlaintext, self.__CaesarCiphertex
t,
        #self.__CaesarShift

    def getVigNoKeyInfo(self):
        return self.__VigNoKeyQueue, self.__VigNoKeyPlaintext, self.__VigNoKeyCiph
ertext,
        #self.__VigNoKeyKey
```

```
    def getRailFenceInfo(self):
        return self.__RailFenceQueue, self.__RailFencePlaintext, self.__RailFenceC
iphertext
    def getUnknownInfo(self):
        return self.__UnknownCipherQueue, self.__UnknownCipherCipherText, self.__U
nknownCipherPlainText,
        #self.__UnknownCipherColumns, self.__UnknownCipherColumnOrder, self.__Unkn
ownCipherReverseOrder
```

Menu Class

```
class Menu:
    #A.D.A.
     #Imports time (potentially for all functions but apparently have to import in
dividually)

    def __init__(self):

        print("***** Cryptography Companion *****\n")
        print("ADA: Hi There! My name's ADA, and I'm here to help you encode, deco
de and crack your ciphers.")
        time.sleep(1)
        print("ADA: We can do many things here, focusing on a few ciphers in parti
cular, with some help for any beginners out there.")
        time.sleep(1)
        tutorialChoice = input("ADA: Would you like to take a look at a Code-
Breaking tutorial? \nYou: ")

        while tutorialChoice == "":
            tutorialChoice = input("ADA: I'm sorry I didn't get that. Can you try
again? \nYou: ")
        if self.PositiveResponse(tutorialChoice):
            print("ADA: Coming right up!")
            time.sleep(1)
            self.tutorialMenu()
            quitChoice = input("ADA: Are you all finished up? There's plenty more
I can do. \nYou: ")
            if self.PositiveResponse(quitChoice):
                print("ADA: Thanks for being here! See you next time!")
                time.sleep(3)
                quit()

        self.listOfChoices()

    def listOfChoices(self):
        import time
        toQuit = False
        while toQuit == False:
            print("ADA: What cipher would you like to work with?")
            time.sleep(1)
            print("ADA: Our options are Caesar, Rail Fence, Substitution, Transpos
ition, Vigenere.")
            time.sleep(1)
            print("ADA: We can also help analyse your text, or there's a Tutorial
for beginners!")
            time.sleep(1)
            print("ADA: Or if you're ready to quit, just say!")
            time.sleep(1)
            cipherChoice = input("You: ")
            decision = self.whichCipher(cipherChoice)
            if decision == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            if decision == "Q":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit()
            if decision == "C":
                self.CaesarMenu()
```

```
            if decision == "S":
                self.SubstitutionMenu()
            if decision == "RF":
                self.RailFenceMenu()
            if decision == "TR":
                self.TranspositionMenu()
            if decision == "A":
                self.AnalysisMenu()
            if decision == "V":
                self.VigenereMenu()
            if decision == "TU":
                self.tutorialMenu()

    def PositiveResponse(self, text):
        correctedtext = self.CorrectText(text)
        #print("Positive Test")
        positives = ["YES", "Y", "OFCOURSE", "YUP", "YEAH", "YAH", "YH", "DONE"] #
#Add MORE
        if correctedtext in positives:
            return True

    def whichCipher(self, text):
        caesaroptions = ["CAESAR", "CAESARCIPHER"]
        railfenceOptions = ["RAIL", "RF", "RAILFENCE"]
        substitutionOptions = ["SUBSTITUTION", "SUB", "ALPHABETICSUBSTITUTION", "A
S"]
        transpositionOptions = ["TRANS", "COLUMNARTRANPOSITION", "TRANSPOSITION",
"CT"]
        analysisOptions = ["ANALYSIS", "ANALYSE", "CALCULATE", "FREQUENCYANALYSIS"
, "A", "HELPANALYSE"]
        vigenereOptions = ["VIG", "VIGENERE", "VIGENERECIPHER", "V"]
        tutorialOptions = ["TUTORIAL", "HELP"]
        unknownOptions = ["UNKNOWN", "COLDCIPHER", "UNKNOWNCIPHER", "CRYPTOGRAPHY"
, "DECRYPT"]
        quitOptions = ["DONE", "FINISHED", "STOP", "END", "EXIT", "QUIT"]
        correctedtext = self.CorrectText(text)
        if correctedtext in caesaroptions:
            return "C"
        elif correctedtext in substitutionOptions:
            return "S"
        elif correctedtext in railfenceOptions:
            return "RF"
        elif correctedtext in transpositionOptions:
            return "TR"
        elif correctedtext in analysisOptions:
            return "A"
        elif correctedtext in vigenereOptions:
            return "V"
        elif correctedtext in tutorialOptions:
            return "TU"
        elif correctedtext in quitOptions:
            return "Q"
        elif correctedtext in unknownOptions:
            return "U"
        else:
            return "ERROR"

    def GetText(self, remove=False):
        text = ""
```

```
        while text == "":
            choice = input("ADA: Now I need your text. Would you like to enter in
a file name to get the text from? \nYou: ")

            if self.PositiveResponse(choice):
                filename = input("ADA: Then I need a file name. Please include the
 relevant extension (e.g. textfile.txt). Do you mind entering it below? \nYou: ")
                try:
                    textFile = open(filename, "r")
                    text = textFile.read()
                    #print(text)
                    textFile.close()
                except:
                    print("ADA: That file didn't open correctly, please try again!
")
            else:
                text = input("ADA: Oh, then I need you to type in your text. Do yo
u mind entering it below? \nYou: ")
            correctedText = self.CorrectText(text)
            letterText = len(list(filter(lambda x: x.isalpha(), text)))
            #print(correctedText, letterText)
            if correctedText.isalnum() and letterText > 0:
                if remove:
                    text = correctedText
                #print(text)
            else:
                text = ""
                print("ADA: Sorry I don't think that was a valid input. Please try
 that again!")
        return text

    @staticmethod
    def CorrectText(text):
        import string
        #Remove spaces
        newtext = (text.upper()).replace(" ", "")
        #Remove all punctuation too...
        newtext = newtext.translate(str.maketrans('', '', string.punctuation))
        return newtext


    def ModeChoice(self, text, encode = False, decode = False, solve = False, colu
mn = False, rails = False, Frequency=False, Index=False, compare=False, every= Fal
se, replace= False):
        decodeOptions = ["DECIPHER", "DECODE", "DECODING"]
        encodeOptions = ["ENCODE", "ENCIPHER",]
        solveOptions = ["AUTOSOLVE", "DOITFORME", "SOLVE"]
        quitOptions = ["DONE", "FINISHED", "STOP", "END", "EXIT", "QUIT"]
        columnOptions = ["SUGGESTCOLUMNS", "COLUMN", "COLUMNS"]
        railOptions = ["RAILS", "PICKRAILS"]
        FreqOptions = ["FREQUENCY", "FREQUENCYANALYSIS", "ANALYSE"]
        IndexOptions = ["IOC", "INDEXOFCOINCIDENCE", "INDEX"]
        compareOptions = ["COMPARISON", "COMPARE", "VALUES"]
        everyOptions = ["EVERY", "COMBOS", "COMBINATIONS"]
        replaceOptions = ["REPLACE", "SUGGEST", "SUGGESTREPLACEMENTS"]

        correctedtext = self.CorrectText(text)
        if correctedtext in decodeOptions:
            if decode:
```

```
                return "DECODE"
            elif correctedtext in encodeOptions:
                if encode:
                    return "ENCODE"
            elif correctedtext in solveOptions:
                if solve:
                    return "SOLVE"
            elif correctedtext in columnOptions:
                if column:
                    return "COLUMN"
            elif correctedtext in railOptions:
                if rails:
                    return "RAILS"
            elif correctedtext in FreqOptions:
                if Frequency:
                    return "FREQ"
            elif correctedtext in IndexOptions:
                if Index:
                    return "IOC"
            elif correctedtext in compareOptions:
                if compare:
                    return "COMPARE"
            elif correctedtext in everyOptions:
                if every:
                    return "EVERY"
            elif correctedtext in replaceOptions:
                if replace:
                    return "REPLACE"
            elif correctedtext in quitOptions:
                return "QUIT"
            else:
                return "ERROR"
            return "ERROR"

        def tutorialMenu(self): ##Done
            #print("Tutorial Menu Test Running")
            print("ADA: Welcome to the Tutorial!")
            ##Choose either caesar/vig/unknown/railfence USING WHICHCIPHER
            selectionCorrect = False
            while selectionCorrect == False:
                print("ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFenc
e and an Unknown cipher.\nADA: For very beginners, I reccommend our Caesar or Rail
Fence tutorials. \nFor more advanced codebreakers, the best place to start would b
e the Vigenere (which uses a crib) and the Unknown Cipher.")
                tutorialchoice = input("ADA: Which cipher would you like to see a tuto
rial of? \nYou: ")
                cipherChosen = self.whichCipher(tutorialchoice)
                if cipherChosen == "C":
                    queue,  plaintext, ciphertext = Tutorial().getCaesarInfo()
                    selectionCorrect = True
                elif cipherChosen == "RF":
                    queue,  plaintext, ciphertext = Tutorial().getRailFenceInfo()
                    selectionCorrect = True
                elif cipherChosen == "U":
                    queue,  plaintext, ciphertext = Tutorial().getUnknownInfo()
                    selectionCorrect = True
                elif cipherChosen == "V":
                    queue,  plaintext, ciphertext = Tutorial().getVigNoKeyInfo()
                    selectionCorrect = True
```

```
            else:
                print("ADA: That doesn't appear to be a cipher in my databanks. Co
uld you try another?")
            pass

        #
        print("ADA: Now, onto the tutorial!")
        #print("ADA: I'll be showing you the tutorial in the format of rubric then
 ciphertext/etc")
        time.sleep(3)
        #Intro
        Intro = queue.pop(0)
        print("ADA: " + str(Intro))
        time.sleep(3)
        #Main Body
        for x in range(0,len(queue)-1):
            pairOfInfo = queue[x]
            print("ADA: " + str(pairOfInfo[0]))
            time.sleep(3)
            print("ADA: " + str(pairOfInfo[1]))
            time.sleep(3)

        #Outro
        Outro = queue[len(queue)-1]
        print("ADA: " + str(Outro))
        time.sleep(3)

        print("ADA: Hopefully that helped clear things up! Have a nice day! Headin
g back to the menu now...")
        time.sleep(2)

    def CaesarMenu(self): ##Done
        import time
        #print("Caesar Menu Test Running")
        print("ADA: We're now in the world of the Caesar Cipher. Here we can encod
e, decode and autosolve it!")
        time.sleep(1)
        toQuit = False
        while toQuit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, encode=True, decode=True, solve=Tru
e)
            if mode == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                toQuit =True
            elif mode == "ENCODE":
                self.C_DecodeEncode(encode=True)
                print("ADA: We're now back in the Caesar Cipher menu. Here we can
encode, decode and autosolve it!")
            elif mode == "DECODE":
                self.C_DecodeEncode(encode=False)
                print("ADA: We're now back in the Caesar Cipher menu. Here we can
encode, decode and autosolve it!")
            elif mode == "SOLVE":
                self.C_AutoSolve()
```

```
                print("ADA: We're now back in the Caesar Cipher menu. Here we can
encode, decode and autosolve it!")
            else:
                pass

    def C_DecodeEncode(self, encode = True):
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText()
        shiftCorrect = False
        while shiftCorrect == False:
            try:
                if encode:
                    shiftChosen = int(input("ADA: What shift would you like to enc
ode it by?  \nYou: "))
                else:
                    shiftChosen = int(input("ADA: What shift would you like to dec
ode it by?  \nYou: "))
                shiftCorrect= True
                if shiftChosen > 26 or shiftChosen < 0:
                    shiftCorrect= False
            except:
                print("ADA: Sorry that shift is not valid. Please try another!")
                shiftCorrect= False
        if encode:
            caesarEncode = Caesar(plaintext=text, shift=shiftChosen)
            caesarEncoded = caesarEncode.encode()
            print("ADA: And here is your encoded Caesar Cipher: ", caesarEncoded,
"\nADA: Back to the Caesar Menu!")
        else:
            caesarDecode = Caesar(ciphertext=text, shift=shiftChosen)
            caesarDecoded = caesarDecode.decode()
            print("ADA: And here is your decoded Caesar Cipher: ", caesarDecoded,
"\nADA: Back to the menu!")

    def C_AutoSolve(self): ##Done
        import time
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText()
        caesarSolve = Caesar(ciphertext=text)
        caesarSolved = caesarSolve.autoSolve()
        print("ADA: And here are your potential Caesar Cipher solutions: ")
        for x in caesarSolved:
            print("ADA:", x)
            time.sleep(0.75)
        print("ADA: Hope this was useful. Now, back to the menu!\n")

    def TranspositionMenu(self): ##Done
        #print("Transposition Menu Test Running")
        import time
        print("ADA: We're now in the world of the Transposition Cipher. Here we ca
n encode, decode and suggest column numbers!")
        time.sleep(1)
        quit = False
        while quit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, encode=True, decode=True, column=Tr
ue)
```

```
            if mode == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit = True
            elif mode == "ENCODE":
                self.TR_EncodeDecode(encode=True)
                print("ADA: We're now back in the Transposition Cipher menu. Here
we can encode, decode and suggest column numbers!")
            elif mode == "DECODE":
                self.TR_EncodeDecode(encode=False)
                print("ADA: We're now back in the Transposition Cipher menu. Here
we can encode, decode and suggest column numbers!")
            elif mode == "COLUMN":
                self.TR_Column()
                print("ADA: We're now back in the Transposition Cipher menu. Here
we can encode, decode and suggest column numbers!")
            else:
                pass

    def TR_EncodeDecode(self, encode= True):
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText(remove=True)
        columnCorrect = False
        transpositionColumnSuggestion = Transposition(ciphertext=text)
        ##here use SuggestColumns to suggest possible columns.
        decision = input("ADA: Would you like me to suggest a potential number of
columns? \nYou: ")
        if self.PositiveResponse(decision):
            columnIdeas = transpositionColumnSuggestion.FindPossibleColumns()
            print("ADA: Here are the factors of the text length. They're the best
numbers to use for a good Transposition Encryption.")
            tempText = ""
            for x in columnIdeas:
                tempText += str(x)
                tempText += ","
            print("ADA: You could try", tempText, "perhaps?")
        while columnCorrect == False:
            try:
                if encode:
                    ColumnChoice = int(input("ADA: How many columns do you want to
 encode with?  \nYou: "))
                else:
                    ColumnChoice = int(input("ADA: How many columns do you want to
 decode with?  \nYou: "))
                columnCorrect= True
                if len(text) % ColumnChoice != 0:
                    if encode:
                        confirmBadColumn = input("ADA: Are you sure you want this
number of columns? It could result in a bad encryption. I recommend that you use a
 multiple of the key length. \nYou: ")
                    else:
                        confirmBadColumn = input("ADA: Are you sure you want this
number of columns? It could result in a bad decryption. I recommend that you use a
 multiple of the key length. \nYou: ")
                    if self.PositiveResponse(confirmBadColumn):
```

```
                        columnCorrect = True
                        pass #Confirms that the user knows this may end badly
                    else:
                        columnCorrect= False
                else:
                    columnCorrect= True
            except:
                print("ADA: Sorry that number of columns is not valid. Please try
another!")
                columnCorrect= False
        ## Input the correct order, in a right way
        OrderOfColumns = []
        tempColumn = 0
        CorrectOrder = False

        print("ADA: I'm going to ask you the order of the columns now, one by one.
 The first number you enter will be the first column (and vice versa).")
        while CorrectOrder == False:
            #print("Start loop")
            try:
                for x in range(0, ColumnChoice):
                    tempColumn = int(input("ADA: Position " + str(x) + " will be?
\nYou: "))
                    OrderOfColumns.append(tempColumn)
                    #print(OrderOfColumns, "Current Order")
                CorrectOrder = True

            except:
                print("ADA: Sorry that order of columns is not valid. Please try a
gain!")
                tempColumn = 0
                OrderOfColumns = []
            for x in range(0,ColumnChoice):
                if x not in OrderOfColumns:
                    CorrectOrder = False
            if CorrectOrder == False:
                print("ADA: You didn't include all the column positions in your an
swer... Please try again!")
                tempColumn = 0
                OrderOfColumns = []

        if encode:
            transpositionEncode = Transposition(plaintext=text, columns=ColumnChoi
ce, order=OrderOfColumns)
            transpositionEncoded = transpositionEncode.encode()
            print("ADA: And here is your encoded Transposition Cipher: ", transpos
itionEncoded, "\nADA: Back to the menu!")
        else:
            transpositionDecode = Transposition(ciphertext=text, columns=ColumnCho
ice, order=OrderOfColumns)
            transpositionDecoded = transpositionDecode.decode()
            print("ADA: And here is your decoded Transposition Cipher: ", transpos
itionDecoded, "\nADA: Back to the menu!")


    def TR_Column(self): ##Done
        #print("DEV WARNING: UNFINISHED - PROCEED AT OWN RISK")
        import time
        print("ADA: Great choice! Just let my cogs whir...")
```

```
        time.sleep(1)
        text = self.GetText()
        transpositionColumnSuggestion = Transposition(ciphertext=text)
        ##here use SuggestColumns to suggest possible columns.
        columnIdeas = transpositionColumnSuggestion.FindPossibleColumns()
        print("ADA: Here are the factors of the text length. They're the best numb
ers to use for a good Transposition Encryption.")
        tempText = ""
        for x in columnIdeas:
            tempText += str(x)
            tempText += ", "
        print("ADA: You could try", tempText, "perhaps?")
        time.sleep(1)
        print("ADA: That's all for this... See you back in the Menu!")
        time.sleep(1)

    def VigenereMenu(self): ##Done
        #print("Vigenere Menu Test Running")
        import time
        print("ADA: We're now in the world of the Vigenere Cipher. Here we can enc
ode, decode!")
        time.sleep(1)
        quit = False
        while quit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, encode=True, decode=True)
            if mode == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit = True
            elif mode == "ENCODE":
                self.V_EncodeDecode(encode=True)
                print("ADA: You're back in the Vigenere menu. Pick something else
to do!")
            elif mode == "DECODE":
                self.V_EncodeDecode(encode=False)
                print("ADA: You're back in the Vigenere menu. Why don't you try so
mething new?")
            else:
                pass
        pass

    def V_EncodeDecode(self, encode=True):
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText()
        keyCorrect = False
        while keyCorrect == False:
            try:
                if encode:
                    keyChosen = input("ADA: What Key would you like to encode it b
y?  \nYou: ")
                else:
                    keyChosen = input("ADA: What Key would you like to decode it b
y?  \nYou: ")
                keyCorrect= True
```

```
                if keyChosen.isalpha():
                    keyCorrect= True
                else:
                    print("ADA: The Key must be only made of characters in the Alp
habet. I bet you know which those are!")
                    keyCorrect= False
                if len(keyChosen) > len(text):
                    print("ADA: Sorry that key is too long. Please try a shorter o
ne! It should be shorter than the text.")
                    keyCorrect= False

            except:
                print("ADA: Sorry that key is not valid. Please try another!")
                keyCorrect= False
        if encode:
            vigEncode = Vigenere(plaintext=text, key=keyChosen)
            vigEncoded = vigEncode.encode()
            print("ADA: And here is your encoded Vigenere Cipher: ", vigEncoded, "
\nADA: Back to the menu!")
        else:
            vigDecode = Vigenere(ciphertext=text, key=keyChosen)
            vigDecoded = vigDecode.decode()
            print("ADA: And here is your decoded Vigenere Cipher: ", vigDecoded, "
\nADA: Back to the menu!")


    def SubstitutionMenu(self): ##Done
        #print("Substitution Menu Test Running")
        import time
        print("ADA: We're now in the world of the Substitution Cipher. Here we can
 encode, decode and suggest replacements!")
        time.sleep(1)
        quit = False
        while quit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, encode=True, decode=True, replace=T
rue)
            if mode == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit = True
            elif mode == "ENCODE":
                self.S_EncodeDecode(encode=True)
                print("ADA: You've returned to the Substitution menu. You can enco
de, decode or suggest replacements.")
            elif mode == "DECODE":
                self.S_EncodeDecode(encode=False)
            elif mode == "REPLACE":
                self.S_Replace()
            else:
                pass
        pass

    def S_EncodeDecode(self, encode=True):
        #Need the respective letters, must be 26
        text = self.GetText()
```

```
        alphabetTemplate = Analysis().alphabet
        alphabet = ""
        SubCorrect = False
        while SubCorrect == False:
            alphabet = ""
            for letter in alphabetTemplate:
                LetterCorrect = False
                while LetterCorrect == False:
                    letterTemp = input(("ADA: What letter would you like to replac
e '" + letter + "'? \nYou: "))
                    if letterTemp.isalpha():
                        if len(letterTemp) ==1 and letterTemp != " ":
                            alphabet += letterTemp.lower()
                            LetterCorrect = True
                    else:
                        print("ADA: That doesn't appear to be a single letter, cou
ld you try something different?")
            SubCorrect = True
            for x in alphabetTemplate:
                if x not in alphabet:
                    SubCorrect = False
            if SubCorrect == False:
                print("ADA: You appear to have some duplicates in here. Can you tr
y that all again?")
        CompleteAlphabet = alphabet + alphabet.upper()
        if encode:
            substitute = Substitution(plaintext=text, cipherbet=CompleteAlphabet)
            subEncoded = substitute.encode()
            print("ADA: Here is your encoded Substitution Cipher: ", subEncoded)
            time.sleep(2)
        else:
            substitute = Substitution(ciphertext=text, cipherbet=CompleteAlphabet)
            subDecoded = substitute.decode()
            print("ADA: Here is your decoded Substitution Cipher: ", subDecoded)
            time.sleep(2)
        print("ADA: All done now. Heading back to the Substitution Menu...")
        time.sleep(2)

    def S_Replace(self): ##Done
        #Suggest the replacements from the ciphertext
        #print("DEV WARNING: UNFINISHED - PROCEED AT OWN RISK")
        import time
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText()
        replace = Substitution(ciphertext=text)
        replacesuggests = replace.SuggestReplacements()
        print("ADA: Here are the suggested replacements of the text. They're likel
y to be slightly inaccurate but will give you a good place to start!")
        for x in replacesuggests:
            print("ADA: You could try " + str(x[0]) + " (English) to go to " + str
(x[1]) + " (text), perhaps?")

        time.sleep(1)
        print("ADA: That's all for this... See you back in the Menu!")
        time.sleep(1)

    def RailFenceMenu(self): #Done
       # print("Rail Fence Menu Test Running")
```

```
        import time
        print("ADA: We're now in the world of the Rail Fence Cipher. Here we can e
ncode, decode and suggest the number of rails!")
        time.sleep(1)
        quit = False
        while quit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, encode=True, decode=True, rails=Tru
e)
            if mode == "ERROR":
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit = True
            elif mode == "ENCODE":
                self.RF_EncodeDecode(encode=True)
                print("ADA: You're back in the RailFence menu, please stay awhile.
 Here you can encode, decode and suggest the number of rails!")
            elif mode == "DECODE":
                self.RF_EncodeDecode(encode=False)
                print("ADA: You're back in the RailFence menu, please stay awhile.
 Here you can encode, decode and suggest the number of rails!")
            elif mode == "RAILS":
                self.RF_Rails()
                print("ADA: You're back in the RailFence menu, please stay awhile.
 Here you can encode, decode and suggest the number of rails!")
            else:
                pass
        pass

    def RF_EncodeDecode(self, encode=True):
        if encode:
            print("ADA: The Rail Fence cipher is complicated to encode, but it's s
imple with a couple of key pieces of information.")
        else:
            print("ADA: The Rail Fence cipher is complicated to decode, but it's s
imple with a couple of key pieces of information.")
        print("ADA: One of these is the number of rails, the other is your text! "
)
        ##Get text
        text = self.GetText(remove=True)
        railCorrect = False
        while railCorrect == False:
            try:
                ##Get rail number (2 or higher) (this is covered by the "in rf pot
entials")
                if encode:
                    railNum = int(input("ADA: How many rails do you want to use to
 encode this with? \nYou: "))
                else:
                    railNum = int(input("ADA: How many rails do you want to use to
 decode this with? \nYou: "))
                ##If rail number in the array returned by GuessRails
                rfRails = RailFence(ciphertext=text, railNo=railNum)
                guess = len(text) #Check all potential rails in case it doesn't ex
ist then
                rfPotentials = rfRails.GuessRails(guess)
```

```
                if railNum in rfPotentials:
                    railCorrect = True
                elif rfPotentials == []:
                    if encode:
                        print("ADA: I'm so sorry but the text you are trying to en
code has no suitable rail numbers to encode with. I'm going to have to ask for you
r text again.")
                    else:
                        print("ADA: I'm so sorry but the text you are trying to de
code has no suitable rail numbers to encode with. I'm going to have to ask for you
r text again.")

                    #### Loops through this until RFPotentials actaully has someth
ing that works. Lets the user know how many extra characters they had to add
                    addChoice = input("ADA: Would you like me to instead add on ch
aracters ('X') to your text (you have just picked) until there is at least one val
id number of rails? \nYou: ")
                    if self.PositiveResponse(addChoice):
                        correctRails = False
                        counter = 0
                        while correctRails == False:
                            counter +=1
                            text += "X"
                            rfRails = RailFence(ciphertext=text, railNo=railNum)
                            rfPotentials = rfRails.GuessRails(len(text))
                            if len(rfPotentials) >= 3:
                                correctRails = True
                        if encode:
                            print("ADA: I have added", counter, "letters to your t
ext. You can now encode it with", rfPotentials)
                        else:
                            print("ADA: I have added", counter, "letters to your t
ext. You can now decode it with", rfPotentials)
                    else:
                        print("ADA: Okay, then you'll have to enter in a different
 piece of text, as that wasn't valid.")
                        text = self.GetText(remove=True)
                else:
                    if encode:

                        print("ADA: In order to properly encode the Rail Fence cip
her, you need a rail number that, when you do the modulus of the length of text by
 ((2*rails) -
2), it should be 1. \nADA: Please try a different number, as this wasn't acceptabl
e.")
                        print("ADA: How about using ", str(rfPotentials), "to enco
de your text with?")
                    else:
                        print("ADA: In order to properly decode the Rail Fence cip
her, you need a rail number that, when you do the modulus of the length of text by
 ((2*rails) -
2), it should be 1. \nADA: Please try a different number, as this wasn't acceptabl
e.")
                        print("ADA: How about using ", str(rfPotentials), "to deco
de your text with?")
            except:
                print("ADA: That doesn't seem right. Could you try something diffe
rent perhaps?")
        if encode:
```

```
            print("ADA: Now I'm going to encode it for you.")
            #print(text)
            railEnc = RailFence(railNo=railNum, plaintext=text)
            railEncoded = railEnc.encode()
            print("ADA: Here is your encoded Rail Fence:", railEncoded)
        else:
            print("ADA: Now I'm going to decode it for you.")
            railDec = RailFence(railNo=railNum, ciphertext=text)
            railDecoded = railDec.decode()
            print("ADA: Here is your decoded Rail Fence:", railDecoded)
        time.sleep(3)
        print("ADA: Heading back to the Rail Fence menu now!")
        time.sleep(1.5)

    def RF_Rails(self): ##Done
        #print("NOT STARTED DON'T CONTINUE.")
        import time
        print("ADA: Great choice! Just let my cogs whir...")
        time.sleep(1)
        text = self.GetText(remove=True)
        print("ADA: I now need for you to input the maximum rails that you want me
 to check up to.")
        time.sleep(1)
        print("ADA: I'd recommend 10 but it's your choice.")
        time.sleep(1)
        guessCorrect = False
        guess = 0
        while guessCorrect == False:
            try:
                guess = int(input("ADA: What would you like it to be then?  \nYou:
 "))
                if guess >2:
                    guessCorrect = True
            except:
                print("ADA: I don't think that was a valid number... Could you try
 something different? Thanks!")
        rfRails = RailFence(ciphertext=text, railNo=5)
        rfPotentials = rfRails.GuessRails(guess)
        tempText = ""
        for x in rfPotentials:
            tempText += str(x)
            tempText += ", "
        print("ADA: And here are your the number of rails you could encode by:", t
empText, "\nADA: Back to the menu!")


    def AnalysisMenu(self): #Done
        #print("Analysis Menu Test Running")
        import time
        print("ADA: We're now in the world of Text Analysis. Here we can display a
 frequency analysis, an Index of Coincidence, compare to the normal English text v
alues, or find combinations of letters!")
        time.sleep(1)
        quit = False
        while quit == False:
            chooseMode = input("ADA: What would you like to do today?  \nYou: ")
            mode = self.ModeChoice(chooseMode, Frequency=True, Index=True, compare
=True, every=True)
            if mode == "ERROR":
```

```
                print("ADA: Sorry I didn't get that.. Perhaps you spelt it wrong?
Try again!")
            elif mode == "QUIT":
                print("ADA: Goodbye!")
                time.sleep(2)
                quit = True

            elif mode == "FREQ":
                self.FrequencyAnalysis()
                print("ADA: You're back in the Analysis menu, what shall you like
to do next? Here we can display a frequency analysis, an Index of Coincidence, com
pare to the normal English text values, or find combinations of letters!")

            elif mode == "IOC":
                self.IndexOfCoincidence()
                print("ADA: You're back in the Analysis menu, what shall you like
to do next? Here we can display a frequency analysis, an Index of Coincidence, com
pare to the normal English text values, or find combinations of letters!")

            elif mode == "COMPARE":
                self.Compare()
                print("ADA: You're back in the Analysis menu, what shall you like
to do next? Here we can display a frequency analysis, an Index of Coincidence, com
pare to the normal English text values, or find combinations of letters!")

            elif mode == "EVERY":
                self.EveryX()
                print("ADA: You're back in the Analysis menu, what shall you like
to do next? Here we can display a frequency analysis, an Index of Coincidence, com
pare to the normal English text values, or find combinations of letters!")

            else:
                pass
        pass

    def FrequencyAnalysis(self): ##Done
        #print("NOT STARTED DON'T CONTINUE.")

        print("ADA: Okay, time to do some Frequency Analysis!")
        time.sleep(1)
        print("ADA: But first...")
        Freqtext = self.GetText()
        print("ADA: Now, using your text, I'll show you the individual frequencies
 for each letter.")
        freq = Analysis(text=Freqtext)
        freqAnalysis = freq.GetCipherFrequencies()
        print("ADA: It'll be 'letter' : value. For example, 'a' : 17.")
        print("ADA: Here we go...")
        for key in freqAnalysis:
            print("ADA: '" + key + "' : ", str(freqAnalysis[key]))
            time.sleep(0.25)

        print("ADA: Maybe that helped with your code-
breaking! Back to the Analysis menu...")

    def IndexOfCoincidence(self): ##Done
        #print("NOT STARTED DON'T CONTINUE.")
```

```
        print("ADA: So... You have chosen to calculate the Index of Coincidence of
 your text.\nThis value tells you how likely it is to get the same letter next in
the piece of text. ")
        time.sleep(1)
        print("ADA: For English, the value is close to 1.73, however for a complet
ely random text, it's close to 1.00.")
        time.sleep(1)
        print("ADA: But before I can do this...")
        time.sleep(0.5)
        ioctext = self.GetText()
        if len(ioctext) < 100:
            print("ADA: WARNING - a short piece of text will have a very inaccurat
e Index Of Coincidence, meaning you might not be able to trust it.")
        time.sleep(1)
        IoC = Analysis(text=ioctext)
        IOCValue = IoC.GetIoCValue()
        IOCValue = round(IOCValue, ndigits= 2)
        print("ADA: The Index Of Coincidence value for this piece of text is:", st
r(IOCValue))
        time.sleep(2)
        print("ADA: Thanks for your time! Back to the Analysis Menu...")
        time.sleep(1)

    def Compare(self): ##Done
        #print("NOT STARTED DON'T CONTINUE.")

        print("ADA: Here, I'm going to show you some comparisons between the text
you 'll enter and the values for a normal English piece of text.")
        time.sleep(1)
        print("ADA: But, before I continue...")
        time.sleep(0.5)
        Perctext = self.GetText()
        print("ADA: I'm going to use the percentages for each letter in the text,
as this is easier to visually compare.")
        perc = Analysis(text=Perctext)
        percAnalysis = perc.GetCipherPercents()
        engPerc = perc.GetEnglishPercents()
        print('''ADA: This'll be in the system:
Letter | English | Ciphertext
----------------------------''')
        for letter in engPerc:
            print( "      " + letter + " |  " + str(engPerc[letter]) + "    | " + st
r(percAnalysis[letter]))

        print("ADA: Hopefully this was useful for you! Heading back to the Analysi
s Menu now...")
        time.sleep(2)

    def EveryX(self): ##Done

        #print("NOT STARTED DON'T CONTINUE.")

        print("ADA: This section can help find common groupings of letters, with t
he number of letters chosen by you! ")
        print("ADA: To explain, if you enter 'abcdefggababcdgf' with 2, it'll calc
ulate the combinations in the text, i.e. 'ab' : 3, 'cd' : 2.")
        print("ADA: But before I can do this...")
        time.sleep(0.5)
        EveryXtext = self.GetText()
```

```
        combo = Analysis(text=EveryXtext)
        print("ADA: The next thing I need is the number of letters per combination
. I usually reccommend 2, but anything above 10 seems excessive.")
        xCorrect = False
        while xCorrect == False:
            try:
                x = int(input("ADA: What value would you like to use? \nYou: "))
                xCorrect = True
                if x > 10:
                    print("ADA: I said that above 10 seems excessive. Please try a
 lower value.")
                    xCorrect = False
                elif x < 0:
                    print("ADA: You cannot have a combination of less than 0 lette
rs. Please try again!")
                    xCorrect = False
            except:
                print("ADA: Hmm that doesn't appear to be an integer. Could you tr
y something different?")
        sortedCombo = combo.MostCommonXLetters(LetterNum=x)
        print("ADA: Now that I've calculated this, how many results would you like
? (E.G. Top 10, top 5...)")
        valueCorrect = False
        value = 0
        while valueCorrect == False:
            try:
                value = int(input("ADA: How many would you like to see? \nYou: "))
                valueCorrect = True
                if value > 100:
                    print("ADA: I think that above 100 seems excessive. Please try
 a lower value.")
                    valueCorrect = False
                elif value < 0:
                    print("ADA: You cannot have a Top X of less than 0 items. Plea
se try again!")
                    valueCorrect = False
            except:
                print("ADA: Hmm that doesn't appear to be an integer. Could you tr
y something different?")
        print("ADA: It'll be 'combo' : key. For example, 'abc' : 17.")
        print("ADA: Here we go...")
        for i in range(0, value):
            #print(i, "test")
            print("ADA: '" + str(sortedCombo[i][0]) + "' : ", str(sortedCombo[i][1
]))
            time.sleep(0.25)

        print("ADA: That's all finished now! Back to the Analysis Menu...")
        time.sleep(2)
```

Analysis Class

```
class Analysis:
    def __init__(self, text = ""):
        import string
        ##Removes spaces
        self.__text = (text.lower()).replace(" ", "")
        ###Removes all punctuation too...
        self.__text = self.__text.translate(str.maketrans('', '', string.punctuati
on))
        #print(x)
        #print(string.punctuation)
        #print(self.__text)

        #DICTIONAIRIES to store the values of the letters/percentages of each lett
er
        self.__EnglishPercents = { "a":  8.04, "b": 1.48, "c": 3.34,"d": 3.82,"e":
 12.49,"f": 2.40,"g": 1.87,"h": 5.05,"i": 7.57,"j": 0.16,"k": 0.54,"l": 4.07,"m":
2.51,"n": 7.23,"o": 7.64,"p": 2.14,"q": 0.12,"r": 6.28,"s": 6.51,"t": 9.28,"u": 2.
73,"v": 1.05,"w": 1.68,"x": 0.23,"y": 1.66,"z": 0.09}
        self.__CipherPercents = {"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.__pairedUp = [] # an ARRAY to store the paired up english to cipherte
xt values (later to be a 2D array)
        self.__CipherFrequencies = {"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} #Empty to put values into

        ##Unused alphabet array
        self.alphabet = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","
o","p","q","r","s","t","u","v","w","x","y","z"]
        self.__iocValue = 0 #a FLOAT
        self.take2 = False

        #list of preordered english values and percents
        # Means reduces time sorting the dictionary each time you instantiate the
class
        self.__orderedEnglishList = [12.49, 9.28, 8.04, 7.64, 7.57, 7.23, 6.51, 6.
28, 5.05, 4.07, 3.82, 3.34, 2.73, 2.51, 2.40, 2.14, 1.87, 1.68, 1.66, 1.48, 1.05,
0.54, 0.23, 0.16, 0.12, 0.09]
        self.__orderedEnglishLetterList = ["e","t","a","o","i","n", "s", "r", "h",
 "l", "d", "c", "u","m", "f", "p", "g", "w", "y", "b", "v", "k", "x", "j", "q", "z
"]
        ## Values From http://www.norvig.com/mayzner.html
        # Testing  -> , result: 8.04
        #print(self.__EnglishPercents[8.04])

        #Calculates the values of each before moving on
        #Prevents errors occuring from empty dictionairies or the objects from not
 actually computing values
        self.IndexOfCoincidence()
        self.CalculateCipherFrequencies()
        self.CalculateCipherPercents()

    def GetText(self): #Returns the text without punctuation and all lowercase
        return self.__text

    def GetCipherPercents(self): #returns the dictionary with the percentages
        return self.__CipherPercents
```

```
    def GetCipherFrequencies(self): #returns the dictionary with the frequencies
        return self.__CipherFrequencies

    def GetEnglishPercents(self):
        return self.__EnglishPercents

    def GetIoCValue(self): #returns the IOC
        return self.__iocValue

    def makeCipherList(self): # makes a list of all the values in cipherpercents
        cipherList = []
        for key in self.__CipherPercents:
            cipherList.append(self.__CipherPercents[key])
        newcipherList = self.SortIt(cipherList, reverse=True)
        return newcipherList

    def returnkeyfrompercvalue(self, search):
        for name, value in self.__CipherPercents.items(): #uses a LINEAR SEARCH to
 return the key from the value
            if value == search:
                return name

    def MostCommonXLetters(self, LetterNum=2): #a sliding count, creates bigrams b
ut in a sliding way
        ## "abcdef" => "ab", "bc", "cd", "de", "ef"

        #Explanation:
        ## Uses a dictionary, for the combos and the occurences { "ab": 1, "bc":5,
 ....}
        ## sort by the occurences, using sorted(dictionary, key=dictioanry value)
--> then reverses it.
        # later will print out combo, ":", value --> for each pairing.
        bigselection = {}
        try:
            for x in range(0, len(self.__text)): #for every letter in the text
                combo = ""
                for digit in range(0, LetterNum): #creates a combo with length spe
cified
                    combo += self.__text[x+digit]
                if combo in bigselection: #if it's already in the dictionary
                    bigselection[combo] +=1 #add one to the value
                else:
                    bigselection.update( {combo : 1} ) #instead, add to dictionary
 with occurrence one
        except:
            pass #If it fails, move on
        #print(bigselection)

        #Sorts the dictionary by value
        ###WRITE OWN SORT?? QUICKSORT??###

        #sortedDict = sorted(dictionary, key=dictionary.get, reverse = True)
        sortedDictionary = self.SortIt(bigselection, take2=True, reverse=True) #so
rts the dictionary using a QUICKSORT and returns a list

        return sortedDictionary
```

```
        ##Objective: Will match up most common groups of 3 letters, i.e. if gji co
mes up a lot, could suggest its "the"

    def PairUpValues(self):
        # The idea here is to match the highest percents with similarly high perce
nts
        # in a 2D array#
        # e.g. A (plaintext) is matched with g (ciphertextt)
        #explanation: makes a list of the values, order them, then use indexes to
match them in a 2d array
        cipherList = self.makeCipherList()
        for x in range(0, len(cipherList)):
            value = cipherList[x] #takes the next value from cipherList
            letter = self.returnkeyfrompercvalue(value) #uses that value to find t
he key in the dictionary
            self.__pairedUp.append([self.__orderedEnglishLetterList[x], letter]) #
append to a 2D array
        return self.__pairedUp

    def CalculateCipherFrequencies(self):
        for key in self.__CipherFrequencies: #for the length of the frequency dict
ionary
            self.__currentletter = key #sets the variable to the current letter

            #Haskell Style!
            number = len(list(filter(lambda x: x==key, self.__text))) #returns the
 length of the list of the filter object of the text
            # Reason for use: not using elsewhere
            #                 (& streamlines the code)
            self.__CipherFrequencies[key] = number #the length is the number of oc
curences in the text
        #print(self.__CipherFrequencies)

    def CalculateCipherPercents(self): #Very similar to CalculateCipherFrequencies
        total = len(self.__text) #the maximum letters it can be
        for key in self.__CipherPercents:
            number = self.__CipherFrequencies[key]
            try:
                percent = round((number / total)*100, 3) #round the percentage to
3 decimal places
            except:
                percent = 0
            #print(percent)
            self.__CipherPercents[key] = percent #adds into the dictionary
        #print(self.__CipherPercents)

    def IndexOfCoincidence(self):
        ## WARNING LESS THAN 50 LETTERS, UNRELIABLE
        #Method from: https://www.thonky.com/kryptos/index-of-coincidence

        ##PhiO
        self.CalculateCipherFrequencies() #calculates frequencies (f)
        F = 0
        phiO = 0
        for key in self.__CipherFrequencies:
            F = (self.__CipherFrequencies[key]) * (self.__CipherFrequencies[key] -
1) #calculate frequencies * (frequencies-1) (F)
            phiO += F # add up all F (phiO)
        #print(phiO)
```

```
        ##PHIR
        N = len(self.__text)#count total letters (excluding spaces) (N)
        #print(N)
        phiR = N * (N-1) * 0.0385 # N * (N-
1) * 0.0385 (this is the random prob) (phiR)
        #print(phiR)

        if phiR > 0:
            ioc = phiO / phiR #Divide phiO by phiR
        else:
            ioc = None #if the length of the text is one, you can't have a valid i
oc
        #print(ioc)
        self.__iocValue = ioc
        return ioc

        #ioc close to 1.00 -> possibly vigenere
        #ioc close to 1.73 -> english (pick closest to ioc 1.73?)

    def SortIt(self, toBeSorted, take2 = False, reverse=False):

        if take2 == False:
            self.take2 = False
        else:
            self.take2 = True
        listOfItems = []
        if type(toBeSorted) is list:
            listOfItems = toBeSorted
        elif type(toBeSorted) is dict:
            for key in toBeSorted:
                listOfItems.append((key,toBeSorted[key])) # Using a TUPLE here
        newList = self.QuickSort(listOfItems)
        if reverse == True:
            newList.reverse()

        return newList

    def QuickSort(self, items):

        #This is a quick sort
        #Using Algorithm from "Edexcel AS and A Level Modular Mathematics D1"
        #By Susie G Jameson (Page 12)

        if len(items) == 1:
            return items
        elif items == []:
            return []
        else:
            LessList = []
            MoreList = []
            #Take an element as the pivot
            #For this we are using the first element
            pivot = items.pop(0)
            #Loop through the remaining items
            for element in items:
                if self.take2: #Because this work with tuples and arrays, must hav
e this
                    if element[1] < pivot[1]:
```

```
                LessList.append(element)
            #If the item is less than the pivot, add to a LessList
                else:
                    MoreList.append(element)
            #If the item is greater than the pivot, add to MoreList
                else:
                    if element < pivot:
                        LessList.append(element)
            #If the item is less than the pivot, add to a LessList
                    else:
                        MoreList.append(element)

            #quicksort on each of these lists
            newLess = self.QuickSort(LessList)
            #print(newLess)
            newMore = self.QuickSort(MoreList)
            #print(newMore)
            #join the results togther
            #import itertools
            newList = newLess + [pivot] + newMore
            #print("Sorted", newList)
            #Return!
            return newList
```

Main Code

```
##Menu!!
menu = Menu()
```

Testing

Test Strategy

For my tests, I will be testing primarily the code breaking and solving tools that I have written, through unit tests. This means I am looking individually the functions in a class that make up the larger overarching function, for example testing ShiftLetter in my Caesar class, in order for my Encode or Decode to work. Throughout my code I have various "levels" of functions: a high-level function is one that returns a finished answer, and this is the overarching guider for the solution (e.g. WalkThrough or Decode); a low-level function is one that may return something back into the high-level function and serves a role in the final output, but only does a small part of the task and will be called in some way by a high-level (e.g. LetterToAscii, ReversePermutation, or TakeSecond).

Later, I will start a Beta Testing Phase, where I will give my code to the Code Breaking Club and ask them to test it with their work on the Cipher Challenge. This will allow me to see how my code is used in practice, and therefore would be effective testing data.

Test Plan

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
1	1a	Caesar Encode returns a correctly shifted text	Caesar("HelloWorld", shift = 5) ➔ Mjqqt Btwqi	PASS	6
1.a	1aiii +iv	ShiftLetter correctly adds the shift to the letter, and returns the new letter (must not be anything outside of the alphabet)	"H" (shift 5) ➔ M	PASS	N/A
2	2a	Caesar Decode returns a correctly shifted (but by negative shift) text	Caesar("MjqqtBtwqi", shift = 5) ➔ HelloWorld	PASS	6
3	4a	Caesar AutoSolve returns all possible outcomes	26 texts returned	PASS	6
4	1d	Railfence Encode returns a correctly encoded text	RailFence(ciphertext="", plaintext="HELPSTUCKINLAIRFREEME!MEG", railNo=4) -> HUAEGETCLIEMELSKNRREMPIF!	PASS	5
4.a	1diii	FinalRail correctly returns the final letters along the zigzag.	RailFence(ciphertext="", plaintext="HELPSTUCKINLAIRFREEME!MEG", railNo=4) -> PIF!	PASS	5

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
4.b	1diii	FirstRail correctly returns only the first row and no extra letters	RailFence(ciphertext="", plaintext="HELPSTUCKINLAIRFREEME!MEG", railNo=4) -> HUAEG	PASS	5
4.b.1	1diii	FirstRail doesn't error if no text is entered	rail = 5, text = "" -> firstrail = "" (no error thrown)	PASS	5
4.c	1diii	MiddleRail correctly returns the two letters specified	RailFence(ciphertext="", plaintext="HELPSTUCKINLAIRFREEME!MEG", railNo=4) and MiddleRail(1,3) -> ETCLIEME	PASS	5
5	2d	Railfence Decode returns a correctly decoded text	RailFence(ciphertext="HUAEGETCLIEMELSKNRREMPIF!", plaintext="", railNo=4) -> HELPSTUCKINLAIRFREEME!MEG	PASS	2
5.a	2diii	Railfence SplitIntoRows correctly return the rows of the railfence in an array	RailFence(ciphertext="HUAEGETCLIEMELSKNRREMPIF!", plaintext="", railNo=4) -> ['HUAEG', 'ETCLIEME', 'LSKNRREM', 'PIF!']	PASS	2
5.b	2diii	RailFence PickingValues correctly takes the right values and returns the decoded railfence	raily.PickingValues(['HUAEG', 'ETCLIEME', 'LSKNRREM', 'PIF!']) -> HELPSTUCKINLAIRFREEME!MEG	PASS	4
5.c	N/A	GuessRails returns the correct numbers of rail that can be encoded by the text	RailFence(ciphertext="HUAEGETCLIEMELSKNRREMPIF!", plaintext="", railNo=4) -> [2, 3, 4, 5, 7]	PASS	3
6	1b	Substitution Encode correctly replaces each of the values in the right places	Substitution(plaintext="There is a stubbornness about me that never can bear to be frightened at the will of others. My courage always rises at every attempt to intimidate me[11].", cipherbet="xcvbnmzasdfghjklqwertyuiopXCVBNMZASDFGHJKLQWERTYUIOP") -> Ranwn se x ertcckwjjnee xcktr hn raxr jnynw vxj cnxw rk cn mwszarnjnb xr ran usgg km kranwe. Ho vktwxzn xguxoe wsene xr nynwo xrrnhlr rk sjrshsbxrn hn.	PASS	7

[11] From Pride and Prejudice – Jane Austen

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
7	2b	Substitution Decode correctly replaces the values in the right place (just an opposite version)	Substitution(ciphertext=" Ranwn se x ertcckwjjnee xcktr hn raxr jnynw vxj cnxw rk cn mwszarnjnb xr ran usgg km kranwe. Ho vktwxzn xguxoe wsene xr nynwo xrrnhlr rk sjrshsbxrn hn.", cipherbet="xcvbnmzasdfghjklqwertyuiopXCVBNMZASDFGHJKLQWERTYUIOP") -> There is a stubbornness about me that never can bear to be frightened at the will of others. My courage always rises at every attempt to intimidate me.	PASS	7
8	4b	Substitution SuggestReplacements returns the pairings of the highest value letters in text VS letters in English	Substitution(plaintext="There is a stubbornness about me that never can bear to be frightened at the will of others. My courage always rises at every attempt to intimidate me.", cipherbet="xcvbnmzasdfghjklqwertyuiopXCVBNMZASDFGHJKLQWERTYUIOP") -> [['e', 'n'], ['t', 'r'], ['a', 'x'], ['o', 'w'], ['i', 'e'], ['n', 'k'], ['s', 'k'], ['r', 'j'], ['h', 'a'], ['l', 'a'], ['d', 'a'], ['c', 'g'], ['u', 'g'], ['m', 'g'], ['f', 'b'], ['p', 'b'], ['g', 'b'], ['w', 'b'], ['y', 'b'], ['b', 'b'], ['v', 'l'], ['k', 'd'], ['x', 'd'], ['j', 'd'], ['q', 'd'], ['z', 'd']]	PASS	7
9	1c	Vigenère Encode correctly adds the key's various values onto the text, missing out the punctuation (if any).	A Correct Vigenere encoding seen. HelloWorld with a key of "key" goes to "RijvsUyvjn"	PASS	39
9.a	1ciii + iv	ShiftLetterUp adds on the value of the key to the text	Shift letter of 'a' and 's' goes to 's'	PASS	39
10	2c	Vigenère Decode correctly minuses the key's various values from the text, missing out the punctuation (if any).	A correct Vigenere encoding seen. "RijvsUyvjn" with a key of 'key' goes to HelloWorld	PASS	39
10.a	2ciii	ShiftLetterDown subtracts on the value of the key to the text	Shift of 's' and 's' goes to 'a'	PASS	39

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
12	1e	Transposition Encode takes the correct letters from the columns	A correct transposition encoding seen. “HELLOWORLD” using [0,1,2,3,4] goes to HWEOLRLLOD	PASS	23
13	2e	Transposition Decode takes the correct letters from the ciphertext	A correct transposition encoding seen. “HWEOLRLLOD” using [0,1,2,3,4] goes to HELLOWORLD	PASS	23
13.a	2eiv	ReversePermutation takes the key, and rearranges it, so each value is in the right place to rearrange the text	An input of [2,0,1] returns as [1,2,0]	PASS	23
14	N/A	FindPossibleColumns suggests factors of the text as potential column numbers	Factors returned by FindPossibleColumns	PASS	23
15	5b+5b ii	Tutorial has all the specified ciphers in a getter	GetCaesarInfo -> returns relevant cipher info	PASS	1
15.a	5bii	Specific good examples plus a description stored as hard-written values	GetCaesarInfo -> returns relevant cipher info	PASS	1
17	3aii1	Analysis of the Text Percentages returns correct percentages of letters	Values seen correct. Example text is “helloworld”	PASS	22
18	3aii2	Analysis of the Text letter values returns correct values of letters	Values seen correct. Example is “helloworld” with h:1, e:1, l:3, o:2, w:1, r:1, d:1	PASS	22
19	3c	Analysis of the IOC returns a correct value for the IOC	Values seen correct. Example is HelloWorld	PASS	22
20	3aiv	PairUpValues matches up letter with the closest percentages to English, depending on the text it can result in vastly different matches.	Values are matched to each letter, doesn’t have to be correct.	PASS	22
21	3aiii	MostCommonXLetters returns an array of tuples containing the most common values	All possible combiantions hown with the number of occurrences beside it	PASS	22
21.a	3aiii	You can change the number of letters used (2, 3, 5 or any other)	Seen in Menu system **(Test No. 44)**, but an example number of 4 input.	PASS	22
22	3b	The English Values are accessible in the main code/accessible outside the Analysis Class	Can call the function GetEnglishPercents outside the class	PASS	22
23	N/A	The menu system lets you access all the ciphers	In the menu, you can access any cipher	PASS	8

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
23.a	N/A	The menu system only accepts valid input when asking for an input	Rejects blank input	PASS	9
23.b	N/A	The Menu System directs you to the right cipher	When “Caesar” input, then takes you to the Caesar menu	PASS	8
23.c	N/A	When erroneous cipher input in Menu, appropriate error message	When “Garble” input, error response	PASS	8
24	N/A	GetText returns only valid texts	It completes all the tests correctly	PASS	N/A
24.a.1	N/A	GetText rejects blank texts (must have at least one letter)	When a blank input, rejected	PASS	10
24.a.2	N/A	Non alphanumerical texts with less than one letter rejected by GetText	If “12345!?” entered, rejected	PASS	10
24.a.3	N/A	When specified, GetText removes all punctuation and returns an all uppercase string	When remove=True, an input of “a…?BcDeF” is returned as “ABCDEF”	PASS	20
24.b	N/A	GetText can open and use a file for you	A file is shown to be opened (no rejection message)	PASS	8
24.b.1	N/A	If you try to open an invalid file, or one doesn’t exist, an appropriate error message is shown	An appropriate error message is shown	PASS	8
24.b.2	N/A	If the text in that file is not valid, it is rejected until an appropriate file is opened or text input.	If the text has no characters in the alphabet, rejected.	PASS	21
25	N/A	The Caesar menu directs you to encode, decode and autosolve	3 options that you can choose (encode, decode, autosolve)	PASS	8
25.a	1a & 2a	C_DecodeEncode takes the correct inputs (text and shift)	Two inputs: text, and shift	PASS	8
25.a.1	1aii & 2aii	C_DecodeEncode rejects shifts greater than 26	When 38 input, rejected	PASS	8
25.a.2	1aii & 2aii	C_DecodeEncode accepts between 0 and 26	When 12 input, accepted	PASS	8
25.a.3	1aii & 2aii	C_DecodeEncode rejects less than 0 shifts	When -5 input, rejected	PASS	8
25.a.4	1aii & 2aii	C_DecodeEncode accepts 26 as a shift	When 26 input, accepted	PASS	11

Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
26	4a	AutoSolve simply asks you to enter a text	Only text input	PASS	10
26.a	4ai	Autosolve prints all 26 possibilities for the text	26 possibilities printed to the screen	PASS	10
27	N/A	Transposition menu has 3 options available, encode, decode and suggest columns	3 options displayed to the user	PASS	14
28	2e	T_EncodeDecode takes the correct inputs	3 inputs: the text, the number of columns and the order of the columns	PASS	12
28.a.1	1eiii & 2eiii	T_EncodeDecode only accepts specific column numbers (if they are generated by suggestColumn)	A correct column number picked that is in the suggested columns	PASS	12
28.a.2	1eiii & 2eiii	T_EncodeDecode only accepts specific column numbers (if the user confirms a bad selection)	The user confirms they want an imperfect number of columns	PASS	12
28.a.3	1eiii & 2eiii	T_EncodeDecode only accepts column orders that have every number in	3 columns, [0,1,3] rejected	PASS	13
28.a.4	1eiii & 2eiii	T_EncodeDecode rejects column orders that have multiples	3 columns, [0,0,2] rejected	PASS	13
28.a.5	1eiii & 2eiii	T_EncodeDecode rejects column orders that have incorrect items in (non-integer values)	3 columns [0,1,b] rejected	PASS	13
28.a.6	1eiii & 2eiii	T_EncodeDecode repeats until a valid column order	Evidence of acceptance when only valid	PASS	13
28.b.1	N/A	If user enters a positive response, then ADA suggest columns to the user	If ‘Yes’ then columns shown	PASS	14
28.b.2	N/A	If the user does anything else but a positive response, then no columns suggested in T_EncodeDecode	If ‘ab34!?’ then no columns shown	PASS	15
29	N/A	TR_Column takes a piece of text and returns the column numbers that can be used as a factor.	A number of columns is printed to the screen appropriately.	PASS	16
30	N/A	The vigenere menu has 2 options to be directed to (encode and decode)	You can go to two places from the vignere menu	PASS	17
31	1c & 2c	V_EncodeDecode takes two inputs, text and the key	2 inputs: text and the key	PASS	17

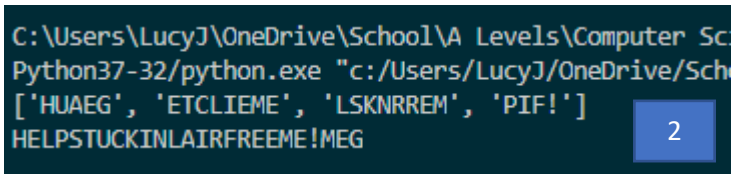
Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
31.a.1	1cii & 2cii	The key must be only alphabet characters, of any case	'Every' accepted	PASS	17
31.a.2	1cii & 2cii	The key will be rejected if it contains any numbers	'Ev3ry' rejected	PASS	17
31.a.3	1cii & 2cii	Key can't be longer than the text	If the text is "hello" a key of "helloworld" is rejected	PASS	19
31.a.4	1cii & 2cii	Key can be the same length as the text	If the text is 'hello', a key of' hello' is accepted	PASS	19
31.a.5	1cii & 2cii	Key can be less than the text	If the text is 'hello', a key of 'key' is accepted	PASS	19
31.a.6	1cii & 2cii	A blank key is rejected	'' rejected	PASS	18
31.a.7	1cii & 2cii	A key of just whitespace is rejected	" " rejected	PASS	18
32	N/A	Substitution menu gives you access to the 3 functions, encode, decode, and suggest replacements.	3 options to go to, encode, decode, or suggest replacements	PASS	24
33	1biii & 2biii	S_EncodeDecode takes an input for every letter	Input seen for every letter	PASS	24
33.a	1b2	S_EncodeDecode only accepts single letters for each input (per letter)	"a" is accepted	PASS	24
33.b	1bii & 2bii	S_EncodeDecode rejects double letter input	"aa" is rejected	PASS	24
33.c	1bii & 2bii	S_EncodeDecode doesn't throw an error if an uppercase letter is input instead	"A" is accepted	PASS	24
33.d	1biv & 2biv	S_EncodeDecode rejects a blank letter input	" " is rejected	PASS	24
33.e	1bv & 2bv	S_EncodeDecode rejects if multiple letters used	When 'a' is used for two inputs, it is rejected	PASS	24 & 25
34	4b	S_Replace takes a text input, and prints out corresponding letters of English and plaintext, (based off percentages)	Pairs of letters are output.	PASS	26
35	N/A	Railfence menu directs you to 3 options, encode decode and suggesting rails.	3 options are available to the user directing to each menu	PASS	27

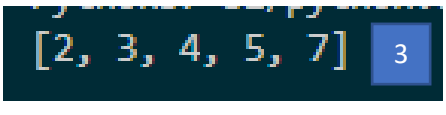
Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
36	1d & 2d	RF_EncodeDecode takes 2 inputs, text and the number of rails	2 types of inputs	PASS	27
36.a	1dii & 2dii	The number of rails must be an integer	“a” rejected	PASS	27
36.b	1dii & 2dii	The number of rails must be a positive integer	“-1” rejected	PASS	27
36.c	1dii & 2dii	The number of rails must be valid for that text (is it produced by GuessRails?)	Appropriate error message	PASS	27
36.d	N/A	If the piece of text doesn’t have any possible rail numbers, ask for a different text input	Appropriate error message returned	PASS	27
37	N/A	RF_Rails takes in a text input and a maximum guess integer.	The 2 inputs seen	PASS	28
37.a	N/A	Maximum guess integer has to be greater than 2	Input of 2 rejected (BOUNDARY)	PASS	28
37.b	N/A	Maximum guess integer has to be an integer	Input of ‘a’ rejected (ERRONEOUS)	PASS	28
37.c	N/A	Maximum guess integer is accepted when it is valid (guess>2)	Input of “10” accepted	PASS	28
37.d	N/A	Rail selection potentials output to the screen	output of rail numbers to the screen	PASS	28
38	N/A	Analysis Menu directs you to 4 options, frequency analysis, index of coincidence, comparison to the english text values, and combinations of letters	4 options available/seen to be accessed	PASS	29
39	3ai	Frequency Analysis takes in only a text input	Text input seen	PASS	29
40	3aii2	Frequency Analysis shows values to the screen	Values seen on screenshot	PASS	29
41	3c	Index Of Coincidence takes in only a text input	Text input seen	PASS	30
42	3c	Index of coincidence shows the IOC value for the text	Value seen on screenshot	PASS	30
43	3ai	Compare takes in a text input	Text input seen	PASS	31
43.a	3bi	Compare shows the English values for each letter	English values seen on screenshot	PASS	31

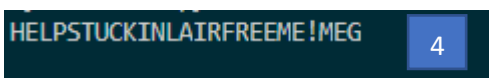
Test Number	Relevant Objective	Description of Test	Expected Outcome	Pass/Fail	Screenshot No.
43.b	3aii1	Compare shows the percentages of each letter in the text	Ciphertext percentages seen on the screenshot	PASS	31
44	3aiii	EveryX takes in a text input and a number of letters to combine, and the maximum number of values to show.	The 3 inputs seen	PASS	32
44.a.1	3aiii	EveryX Guess only takes integer inputs	3 is accepted	PASS	34
44.a.2	3aiii	EveryX Guess rejects float inputs	0.34 is rejected	PASS	32
44.a.3	3aiii	EveryX Guess rejects alphabet inputs	“abc” is rejected	PASS	32
44.a.4.i	3aiii	Every X Guess rejects inputs less than 0	-2 is rejected	FAIL	32
44.a.4.ii	3aiii	EveryX Guess rejects inputs less than 0	-2 is rejected	PASS	34
44.b.1	N/A	EveryX Top Number only takes integer inputs	3 is accepted	PASS	34
44.b.2	N/A	EveryX Top Number rejects float inputs	0.34 is rejected	PASS	33
44.b.3	N/A	EveryX Top Number rejects alphabet inputs	“abc” is rejected	PASS	33
44.b.4	N/A	Every X Top Number rejects inputs less than 0	-2 is rejected	PASS	33
44.b.5	N/A	EveryX Top Number rejects inputs less than 0	-2 is rejected	FAIL	34
45	5bii	Tutorial Menu give you access to 4 tutorials	4 options shown/are visible	PASS	35
45.a	5bii	Directs you to caesar tutorial when required	Input of Caesar accepted & taken to caesar	PASS	36
45.b	5bii	Directs you to railfence tutorial when required	Input of “railfence” accepted and taken to Railfence	PASS	37
45.c	5bii	Directs you to Vigenere tutorial when required	Input of “vigenere” accepted and taken to Vigenere	PASS	38
45.d	5bii	Directs you to Unknown cipher tutorial when required	Input of “Unknown” accepted and taken to Unknown cipher	PASS	39
45.e	5bii	Any other inputs in the tutorial menu returns an appropriate error message (ERRONEOUS)	Input of blargh rejected	PASS	35
45.f	5bii	Any other inputs in the tutorial menu returns an appropriate error message (ERRONEOUS)	Input of 12345!? rejected	PASS	35
45.g	5bii	Any other inputs in the tutorial menu returns an appropriate error message (BOUNDARY)	Input of Substitution rejected	PASS	35

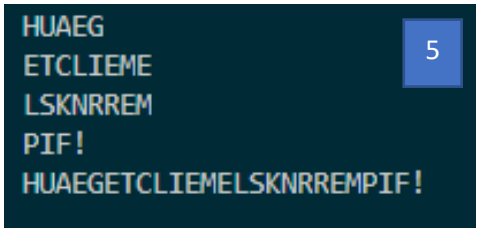
Screenshots

The numbers in the testing table refer to the numbers on the screenshots. Look through to find the one you're looking for!









```
MjqqtBtwqi
HelloWorld
['MjqqtBtwqi', 'LippsAsvph', 'KhoorZruog', 'JgnnqYqtnf', 'IfmmpXpsme', 'HelloWorld', 'GdkknVnqkc', 'FcjjmUmpjb',
'EbiilTloia', 'DahhkSknhz', 'CzggjRjmgy', 'ByffiQilfx', 'AxeehPhkew', 'ZwddgOgjdv', 'YvccfNficu', 'XubbeMehbt', '
WtaadLdgas', 'VszzcKcfzr', 'UryybJbeyq', 'TqxxaIadxp', 'SpwwzHzcwo', 'RovvyGybvn', 'QnuuxFxaum', 'PmttwEwztl', 'O
lssvDvysk', 'NkrruCuxrj']
```

6

```
[['e', 'n'], ['t', 'r'], ['a', 'x'], ['o', 'w'], ['i', 'e'], ['n', 'k'], ['s', 'k'], ['r', 'j'], ['h', 'a'], ['l'
, 'a'], ['d', 'a'], ['c', 'g'], ['u', 'g'], ['m', 'g'], ['f', 'b'], ['p', 'b'], ['g', 'b'], ['w', 'b'], ['y', 'b'
], ['b', 'b'], ['v', 'l'], ['k', 'd'], ['x', 'd'], ['j', 'd'], ['q', 'd'], ['z', 'd']]
Ranwn se x ertcckwjjnee xcktr hn raxr jnynw vxj cnxw rk cn mwszarnjnb xr ran usgg km kranwe. Ho vktwxzn xguxoe ws
ene xr nynwo xrrnhlr rk sjrshsbxrn hn.
There is a stubbornness about me that never can bear to be frightened at the will of others. My courage always ri
ses at every attempt to intimidate me.
```

7

```
***** Cryptography Companion *****

ADA: Hi There! My name's ADA, and I'm here to help you encode, decode and crack your
ciphers.
ADA: We can do many things here, focusing on a few ciphers in particular, with some h
elp for any beginners out there.
ADA: Would you like to take a look at a Code-Breaking tutorial?
You: no
ADA: What cipher would you like to work with?
ADA: Our options are Caesar, Rail Fence, Substitution, Transposition, Vigenere.
ADA: We can also help analyse your text, or there's a Tutorial for beginners!
ADA: Or if you're ready to quit, just say!
You: Garble
ADA: Sorry I didn't get that.. Perhaps you spelt it wrong? Try again!
ADA: What cipher would you like to work with?
ADA: Our options are Caesar, Rail Fence, Substitution, Transposition, Vigenere.
ADA: We can also help analyse your text, or there's a Tutorial for beginners!
ADA: Or if you're ready to quit, just say!
You: Caesar
ADA: We're now in the world of the Caesar Cipher. Here we can encode, decode and auto
solve it!
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file name to get the text fro
m?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.tx
t). Do you mind entering it below?
You: blargle.fre
ADA: That file didn't open correctly, please try again!
ADA: Sorry I don't think that was a valid input. Please try that again!
ADA: Now I need your text. Would you like to enter in a file name to get the text fro
m?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.tx
t). Do you mind entering it below?
You: Orwell.txt
ADA: What shift would you like to encode it by?
You: 38
ADA: What shift would you like to encode it by?
You: -5
ADA: What shift would you like to encode it by?
You: 12
ADA: And here is your encoded Caesar Cipher:  Uf ime m ndustf oaxp pmk uz Mbdux, mzp
ftq oxaowe iqdq efduwuzs ftudfqqz. Iuzefaz Eyuft, tue otuz zgllxqp uzfa tue ndqmef uz
 mz qrradf fa qeombq ftq huxq iuzp, exubbqp cguowxk ftdagst ftq sxmee paade ar Huofad
k Ymzeuaze, ftagst zaf cguowxk qzagst fa bdqhqzf m eiudx ar sduffk pgef rday qzfqduzs
 mxazs iuft tuy. Ftq tmxximk eyqxf ar nauxqp omnnmsq mzp axp dms ymfe. Mf azq qzp ar
uf m oaxagdqp baefqd, faa xmdsq rad uzpaad puebxmk, tmp nqqz fmowqp fa ftq imxx. Uf p
```

8

```
ADA: What cipher would you like to work with?
ADA: Our options are Caesar, Rail Fence, Substitution, Transposition, Vigenere.
ADA: We can also help analyse your text, or there's a Tutorial for beginners!
ADA: Or if you're ready to quit, just say!
You:
ADA: Sorry I didn't get that.. Perhaps you spelt it wrong? Try again!
ADA: What cipher would you like to work with?
```

9

```
ADA: Oh, then I need you to type in your text. Do you mind en
tering it below?
You:
ADA: Sorry I don't think that was a valid input. Please try that again!
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: 12345!?
ADA: Sorry I don't think that was a valid input. Please try that again!
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: Hello World
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: Hello World
ADA: And here are your potential Caesar Cipher solutions:
ADA: Hello World
ADA: Gdkkn Vnqkc
ADA: Fcjjm Umpjb
ADA: Ebiil Tloia
ADA: Dahhk Sknhz
ADA: Czggj Rjmgy
ADA: Byffi Qilfx
ADA: Axeeh Phkew
ADA: Zwddg Ogjdv
ADA: Yvccf Nficu
ADA: Xubbe Mehbt
ADA: Wtaad Ldgas
ADA: Vszzc Kcfzr
ADA: Uryyb Jbeyq
ADA: Tqxxa Iadxp
ADA: Spwwz Hzcwo
ADA: Rovvy Gybvn
ADA: Qnuux Fxaum
ADA: Pmttw Ewztl
ADA: Olssv Dvysk
ADA: Nkrru Cuxrj
ADA: Mjqqt Btwqi
ADA: Lipps Asvph
ADA: Khoor Zruog
ADA: Jgnnq Yqtnf
ADA: Ifmmp Xpsme
ADA: Hope this was useful. Now, back to the menu!
```

10

```
ADA: What shift would you like to encode it by?
You: 26
ADA: And here is your encoded Caesar Cipher:  Goodbye
ADA: Back to the Caesar Menu!
ADA: We're now back in the Caesar Cipher menu. Here we can encode, decode and autosolve it!
ADA: What would you like to do today?
```

11

12

```
ADA: Would you like me to suggest a potential number of columns?
You: yes
ADA: Here are the factors of the text length. They're the best numbers to use for
a good Transposition Encryption.
ADA: You could try 1,2,3,6,4691,9382,14073,28146, perhaps?
ADA: How many columns do you want to encode with?
You: 4
ADA: Are you sure you want this number of columns? It could result in a bad encryp
tion. I recommend that you use a multiple of the key length.
You: no
ADA: How many columns do you want to encode with?
You: 3
```

13

```
ADA: I'm going to ask you the order of the columns now,
er you enter will be the first column (and vice versa).
ADA: Position 0 will be?
You: 0
ADA: Position 1 will be?
You: 1
ADA: Position 2 will be?
You: 3
ADA: You didn't include all the column positions in you
n!
ADA: Position 0 will be?
You: 0
ADA: Position 1 will be?
You: 0
ADA: Position 2 will be?
You: 2
ADA: You didn't include all the column positions in you
n!
ADA: Position 0 will be?
You: 0
ADA: Position 1 will be?
You: 1
ADA: Position 2 will be?
You: b
ADA: Sorry that order of columns is not valid. Please t
ADA: You didn't include all the column positions in you
n!
ADA: Position 0 will be?
You: 0
ADA: Position 1 will be?
You: 1
ADA: Position 2 will be?
You: 2
ADA: And here is your encoded Transposition Cipher:  IA
OSETAFROCEELISPDILHUTGSOSVTYNOTUNQCYOHPVTWLGTDTONRGOWHM
DRSAAETKTHAIECDMYERUAMEAMRITFEANAUOYVIAABCOTHNUEYNOFTEI
NTENHLTCRNAUFUNAITUIAAOHCODVNERIFHEEHLWSELHUNITWWTRNEDD
```

14

```
ADA: We're now back in the Transposition Cipher menu. Here we can encode, decode and suggest column numbers!
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: HelloThere
ADA: Would you like me to suggest a potential number of columns?
You: yes
ADA: Here are the factors of the text length. They're the best numbers to use for a good Transposition Encryption.
ADA: You could try 1,2,5,10, perhaps?
ADA: How many columns do you want to encode with?
You: 2
```

15

```
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: HLOHRELTEE
ADA: Would you like me to suggest a potential number of columns?
You: ab34!?
ADA: How many columns do you want to decode with?
You: 2
ADA: I'm going to ask you the order of the columns now, one by one. The first number you enter will be the first column (and vice ver
sa).
ADA: Position 0 will be?
You: 0
ADA: Position 1 will be?
You: 1
ADA: And here is your decoded Transposition Cipher:  HELLOTHERE
```

16

```
ADA: What would you like to do today?
You: columns
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do you mind entering it below?
You: Orwell.txt
ADA: Here are the factors of the text length. They're the best numbers to use for a good Transposition Encryption.
ADA: You could try 1, 2, 3, 4, 6, 8, 12, 16, 23, 24, 32, 46, 48, 64, 69, 92, 96, 128, 138, 184, 192, 256, 276, 368, 384,
512, 552, 736, 768, 1104, 1472, 1536, 2208, 2944, 4416, 5888, 8832, 11776, 17664, 35328,  perhaps?
ADA: That's all for this... See you back in the Menu!
ADA: We're now back in the Transposition Cipher menu. Here we can encode, decode and suggest column numbers!
ADA: What would you like to do today?
You:
```

17

```
ADA: We're now in the world of the Vigenere Cipher. Here we can
 encode, decode!
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant exten
sion (e.g. textfile.txt). Do you mind entering it below?
You: Orwell.txt
ADA: What Key would you like to encode it by?
You: Ev3ry
ADA: The Key must be only made of characters in the Alphabet. I
 bet you know which those are!
ADA: What Key would you like to encode it by?
You: Every
ADA: And here is your encoded Vigenere Cipher:  Mo arq e wvzelo
 gfjh yep gr Vtigp, vru rlz gcmgfw ncvz wkpmfmee xcmirizr. Ngrn
xfl Whmkf, ldw tfmi rlxdgiu gros ygw wvvywo me yr zjwmvo xf cwx
```

```
ADA: Back to the menu!
ADA: You're back in the Vigenere menu. Pick something else to d
o!
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant exten
sion (e.g. textfile.txt). Do you mind entering it below?
You: Orwell.txt
ADA: What Key would you like to encode it by?
You:
ADA: The Key must be only made of characters in the Alphabet. I
 bet you know which those are!
ADA: What Key would you like to encode it by?
You:
ADA: The Key must be only made of characters in the Alphabet. I
 bet you know which those are!
```

18

19

```
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind ente
ring it below?
You: hello
ADA: What Key would you like to encode it by?
You: helloworld
ADA: Sorry that key is too long. Please try a shorter one! It s
hould be shorter than the text.
ADA: What Key would you like to encode it by?
You: hello
ADA: And here is your encoded Vigenere Cipher:  oiwwc
ADA: Back to the menu!
ADA: You're back in the Vigenere menu. Pick something else to d
o!
ADA: What would you like to do today?
You: encode
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind ente
ring it below?
You: hello
ADA: What Key would you like to encode it by?
You: key
ADA: And here is your encoded Vigenere Cipher:  rijvs
ADA: Back to the menu!
ADA: You're back in the Vigenere menu. Pick something else to d
o!
ADA: What would you like to do today?
You:
```

```
ADA: Now I need your text. Would you like to enter in a file name to get the text fro
m?
You: no
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: a...?BcDeF
ABCDEF
```

20

```
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant exten
sion (e.g. textfile.txt). Do you mind entering it below?
You: badpunC.txt
ADA: Sorry I don't think that was a valid input. Please try tha
t again!
ADA: Now I need your text. Would you like to enter in a file na
me to get the text from?
You:
```

21

```
{'a': 0.0, 'b': 0.0, 'c': 0.0, 'd': 10.0, 'e': 10.0, 'f': 0.0,
'g': 0.0, 'h': 10.0, 'i': 0.0, 'j': 0.0, 'k': 0.0, 'l': 30.0, '
m': 0.0, 'n': 0.0, 'o': 20.0, 'p': 0.0, 'q': 0.0, 'r': 10.0, 's
': 0.0, 't': 0.0, 'u': 0.0, 'v': 0.0, 'w': 10.0, 'x': 0.0, 'y':
 0.0, 'z': 0.0}
{'a': 0, 'b': 0, 'c': 0, 'd': 1, 'e': 1, 'f': 0, 'g': 0, 'h': 1
, 'i': 0, 'j': 0, 'k': 0, 'l': 3, 'm': 0, 'n': 0, 'o': 2, 'p':
0, 'q': 0, 'r': 1, 's': 0, 't': 0, 'u': 0, 'v': 0, 'w': 1, 'x':
 0, 'y': 0, 'z': 0}
{'a': 8.04, 'b': 1.48, 'c': 3.34, 'd': 3.82, 'e': 12.49, 'f': 2
.4, 'g': 1.87, 'h': 5.05, 'i': 7.57, 'j': 0.16, 'k': 0.54, 'l':
 4.07, 'm': 2.51, 'n': 7.23, 'o': 7.64, 'p': 2.14, 'q': 0.12, '
r': 6.28, 's': 6.51, 't': 9.28, 'u': 2.73, 'v': 1.05, 'w': 1.68
, 'x': 0.23, 'y': 1.66, 'z': 0.09}
2.308802308802309
[['e', 'l'], ['t', 'o'], ['a', 'd'], ['o', 'd'], ['i', 'd'], ['
n', 'd'], ['s', 'd'], ['r', 'a'], ['h', 'a'], ['l', 'a'], ['d',
 'a'], ['c', 'a'], ['u', 'a'], ['m', 'a'], ['f', 'a'], ['p', 'a
'], ['g', 'a'], ['w', 'a'], ['y', 'a'], ['b', 'a'], ['v', 'a'],
 ['k', 'a'], ['x', 'a'], ['j', 'a'], ['q', 'a'], ['z', 'a']]
[('orld', 1), ('worl', 1), ('owor', 1), ('lowo', 1), ('llow', 1
), ('ello', 1), ('hell', 1)]
```

22

```
[1, 2, 5, 10]
[1, 2, 0]
HWEOLRLLOD
HELLOWORLD
```

23

```
ADA: We're now in the world of the Substitution Cipher. Here we can encode, decode and sugg
est replacements!
ADA: What would you like to do today?
You: encode
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: What letter would you like to replace 'a'?
You: aa
ADA: What letter would you like to replace 'a'?
You:
ADA: That doesn't appear to be a single letter, could you try something different?
ADA: What letter would you like to replace 'a'?
You: A
ADA: What letter would you like to replace 'b'?
You: a
ADA: What letter would you like to replace 'c'?
You: c
```

24

```
ADA: What letter would you like to replace 'v'?
You: v
ADA: What letter would you like to replace 'w'?
You: w
ADA: What letter would you like to replace 'x'?
You: x
ADA: What letter would you like to replace 'y'?
You: y
ADA: What letter would you like to replace 'z'?
You: z
ADA: You appear to have some duplicates in here. Can you try that all again?
ADA: What letter would you like to replace 'a'?
```

25

```
ADA: What would you like to do today?
You: suggest
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: Here are the suggested replacements of the text. They're likely to be slightly inaccur
ate but will give you a good place to start!
ADA: You could try e (English) to go to e (text), perhaps?
ADA: You could try t (English) to go to t (text), perhaps?
ADA: You could try a (English) to go to a (text), perhaps?
ADA: You could try o (English) to go to o (text), perhaps?
ADA: You could try i (English) to go to i (text), perhaps?
ADA: You could try n (English) to go to n (text), perhaps?
ADA: You could try s (English) to go to h (text), perhaps?
ADA: You could try r (English) to go to s (text), perhaps?
ADA: You could try h (English) to go to r (text), perhaps?
ADA: You could try l (English) to go to d (text), perhaps?
ADA: You could try d (English) to go to l (text), perhaps?
ADA: You could try c (English) to go to w (text), perhaps?
ADA: You could try u (English) to go to u (text), perhaps?
ADA: You could try m (English) to go to c (text), perhaps?
ADA: You could try f (English) to go to m (text), perhaps?
ADA: You could try p (English) to go to f (text), perhaps?
ADA: You could try g (English) to go to g (text), perhaps?
ADA: You could try w (English) to go to p (text), perhaps?
ADA: You could try y (English) to go to b (text), perhaps?
ADA: You could try b (English) to go to y (text), perhaps?
ADA: You could try v (English) to go to v (text), perhaps?
ADA: You could try k (English) to go to k (text), perhaps?
ADA: You could try x (English) to go to x (text), perhaps?
ADA: You could try j (English) to go to z (text), perhaps?
ADA: You could try q (English) to go to j (text), perhaps?
ADA: You could try z (English) to go to q (text), perhaps?
ADA: That's all for this... See you back in the Menu!
ADA: What would you like to do today?
```

26

```
ADA: We're now in the world of the Rail Fence Cipher. Here we can encode, decode and sugges
t the number of rails!
ADA: What would you like to do today?
You: encode
ADA: The Rail Fence cipher is complicated to encode, but it's simple with a couple of key p
ieces of information.
ADA: One of these is the number of rails, the other is your text!
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: How many rails do you want to use to encode this with?
You: a
ADA: That doesn't seem right. Could you try something different perhaps?
ADA: How many rails do you want to use to encode this with?
You: -1
ADA: That doesn't seem right. Could you try something different perhaps?
ADA: How many rails do you want to use to encode this with?
You: 4
ADA: I'm so sorry but the text you are trying to encode has no suitable rail numbers to enc
ode with. I'm going to have to ask for your text again.
ADA: Would you like me to instead add on characters ('X') to your text (you have just picke
d) until there is at least one valid number of rails?
You: no
ADA: Okay, then you'll have to enter in a different piece of text, as that wasn't valid.
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: Helloworld
ADA: How many rails do you want to use to encode this with?
You: 3
```

27

```
ADA: You're back in the RailFence menu, please stay awhile. Here you can encode, decode and
 suggest the number of rails!
ADA: What would you like to do today?
You: rails
ADA: Great choice! Just let my cogs whir...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell copy.txt
ADA: I now need for you to input the maximum rails that you want me to check up to.
ADA: I'd recommend 10 but it's your choice.
ADA: What would you like it to be then?
You: 2
ADA: What would you like it to be then?
You: a
ADA: I don't think that was a valid number... Could you try something different? Thanks!
ADA: What would you like it to be then?
You: 10
ADA: And here are your the number of rails you could encode by: 2, 4,
ADA: Back to the menu!
```

28

```
ADA: We're now in the world of Text Analysis. Here we can display a frequency analysis, an
Index of Coincidence, compare to the normal English text values, or find combinations of le
tters!
ADA: What would you like to do today?
You: frequency
ADA: Okay, time to do some Frequency Analysis!
ADA: But first...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: Now, using your text, I'll show you the individual frequencies for each letter.
ADA: It'll be 'letter' : value. For example, 'a' : 17.
ADA: Here we go...
ADA: 'a' :  2203
ADA: 'b' :  514
ADA: 'c' :  755
ADA: 'd' :  1181
ADA: 'e' :  3556
ADA: 'f' :  648
ADA: 'g' :  603
ADA: 'h' :  1841
ADA: 'i' :  1968
ADA: 'j' :  24
ADA: 'k' :  215
ADA: 'l' :  1090
ADA: 'm' :  654
ADA: 'n' :  1912
ADA: 'o' :  2068
ADA: 'p' :  528
ADA: 'q' :  22
ADA: 'r' :  1592
ADA: 's' :  1745
ADA: 't' :  2606
ADA: 'u' :  776
ADA: 'v' :  265
ADA: 'w' :  790
ADA: 'x' :  42
ADA: 'y' :  494
ADA: 'z' :  27
ADA: Maybe that helped with your code-breaking! Back to the Analysis menu...
ADA: You're back in the Analysis menu, what shall you like to do next? Here we can display
a frequency analysis, an Index of Coincidence, compare to the normal English text values, o
r find combinations of letters!
ADA: What would you like to do today?
```

29

```
ADA: What would you like to do today?
You: index
ADA: So... You have chosen to calculate the Index of Coincidence of your text.
This value tells you how likely it is to get the same letter next in the piece of text.
ADA: For English, the value is close to 1.73, however for a completely random text, it's cl
ose to 1.00.
ADA: But before I can do this...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: The Index Of Coincidence value for this piece of text is: 1.69
ADA: Thanks for your time! Back to the Analysis Menu...
```

30

```
ADA: What would you like to do today?
You: compare
ADA: Here, I'm going to show you some comparisons between the text you 'll enter and the va
lues for a normal English piece of text.
ADA: But, before I continue...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: I'm going to use the percentages for each letter in the text, as this is easier to vis
ually compare.
ADA: This'll be in the system:
Letter | English | Ciphertext
----------------------------
    a |  8.04   | 7.827
    b |  1.48   | 1.826
    c |  3.34   | 2.682
    d |  3.82   | 4.196
    e |  12.49   | 12.634
    f |  2.4   | 2.302
    g |  1.87   | 2.142
    h |  5.05   | 6.541
    i |  7.57   | 6.992
    j |  0.16   | 0.085
    k |  0.54   | 0.764
    l |  4.07   | 3.873
    m |  2.51   | 2.324
    n |  7.23   | 6.793
    o |  7.64   | 7.347
    p |  2.14   | 1.876
    q |  0.12   | 0.078
    r |  6.28   | 5.656
    s |  6.51   | 6.2
    t |  9.28   | 9.259
    u |  2.73   | 2.757
    v |  1.05   | 0.942
    w |  1.68   | 2.807
    x |  0.23   | 0.149
    y |  1.66   | 1.755
    z |  0.09   | 0.096
ADA: Hopefully this was useful for you! Heading back to the Analysis Menu now...
```

31

```
ADA: What would you like to do today?
You: combinations
ADA: This section can help find common groupings of letters, with the number of letters cho
sen by you!
ADA: To explain, if you enter 'abcdefggababcdgf' with 2, it'll calculate the combinations i
n the text, i.e. 'ab' : 3, 'cd' : 2.
ADA: But before I can do this...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: yes
ADA: Then I need a file name. Please include the relevant extension (e.g. textfile.txt). Do
 you mind entering it below?
You: Orwell.txt
ADA: The next thing I need is the number of letters per combination. I usually reccommend 2
, but anything above 10 seems excessive.
ADA: What value would you like to use?
You: 0.34
ADA: Hmm that doesn't appear to be an integer. Could you try something different?
ADA: What value would you like to use?
You: abc
ADA: Hmm that doesn't appear to be an integer. Could you try something different?
ADA: What value would you like to use?
You: -2
ADA: Now that I've calculated this, how many results would you like? (E.G. Top 10, top 5...
)
```

32

```
ADA: Now that I've calculated this, how many results would you like? (E.G. Top 10, top 5...
)
ADA: How many would you like to see?
You: 0.34
ADA: Hmm that doesn't appear to be an integer. Could you try something different?
ADA: How many would you like to see?
You: abc
ADA: Hmm that doesn't appear to be an integer. Could you try something different?
ADA: How many would you like to see?
You: -2
ADA: It'll be 'combo' : key. For example, 'abc' : 17.
ADA: Here we go...
ADA: That's all finished now! Back to the Analysis Menu...
```

33

```
You: combinations
ADA: This section can help find common groupings of letters, with the number of letters cho
sen by you!
ADA: To explain, if you enter 'abcdefggababcdgf' with 2, it'll calculate the combinations i
n the text, i.e. 'ab' : 3, 'cd' : 2.
ADA: But before I can do this...
ADA: Now I need your text. Would you like to enter in a file name to get the text from?
You: no
ADA: Oh, then I need you to type in your text. Do you mind entering it below?
You: HELLOOWOORRLLDD
ADA: The next thing I need is the number of letters per combination. I usually reccommend 2
, but anything above 10 seems excessive.
ADA: What value would you like to use?
You: -2
ADA: You cannot have a combination of less than 0 letters. Please try again!
ADA: What value would you like to use?
You: 3
ADA: Now that I've calculated this, how many results would you like? (E.G. Top 10, top 5...
)
ADA: How many would you like to see?
You: -2
ADA: You cannot have a Top X of less than 0 items. Please try again!
ADA: How many would you like to see?
You: 3
ADA: It'll be 'combo' : key. For example, 'abc' : 17.
ADA: Here we go...
ADA: 'ldd' :  1
ADA: 'lld' :  1
ADA: 'rll' :  1
ADA: That's all finished now! Back to the Analysis Menu...
ADA: You're back in the Analysis menu, what shall you like to do next? Here we can display
a frequency analysis, an Index of Coincidence, compare to the normal English text values, o
r find combinations of letters!
```

34

```
ADA: Welcome to the Tutorial!
ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.
ADA: For very beginners, I reccommend our Caesar or RailFence tutorials.
For more advanced codebreakers, the best place to start would be the Vigenere (which uses a
 crib) and the Unknown Cipher.
ADA: Which cipher would you like to see a tutorial of?
You: blargh
ADA: That doesn't appear to be a cipher in my databanks. Could you try another?
ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.
ADA: For very beginners, I reccommend our Caesar or RailFence tutorials.
For more advanced codebreakers, the best place to start would be the Vigenere (which uses a
 crib) and the Unknown Cipher.
ADA: Which cipher would you like to see a tutorial of?
You: 12345!?
ADA: That doesn't appear to be a cipher in my databanks. Could you try another?
ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.
ADA: For very beginners, I reccommend our Caesar or RailFence tutorials.
For more advanced codebreakers, the best place to start would be the Vigenere (which uses a
 crib) and the Unknown Cipher.
ADA: Which cipher would you like to see a tutorial of?
You: Substitution
ADA: That doesn't appear to be a cipher in my databanks. Could you try another?
ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.
ADA: For very beginners, I reccommend our Caesar or RailFence tutorials.
For more advanced codebreakers, the best place to start would be the Vigenere (which uses a
 crib) and the Unknown Cipher.
ADA: Which cipher would you like to see a tutorial of?
```

35

```
ADA: Which cipher would you like to see a tutorial of?
You: caesar
ADA: Now, onto the tutorial!
ADA: You start by taking the first letter of the ciphertext. This then needs to be shifted
 back down the alphabet by the shift you used originally.
```

36

```
ADA: Welcome to the Tutorial!
ADA: In total we have 4 tutorials: Caesar, Vigenere, a RailFence and an Unknown cipher.
ADA: For very beginners, I reccommend our Caesar or RailFence tutorials.
For more advanced codebreakers, the best place to start would be the Vigenere (which uses a
 crib) and the Unknown Cipher.
ADA: Which cipher would you like to see a tutorial of?
You: railfence
ADA: Now, onto the tutorial!
ADA: This cipher is one of the most difficult to solve, as it looks on paper to be a transp
osition, however the way you unravel it is vastly different.
```

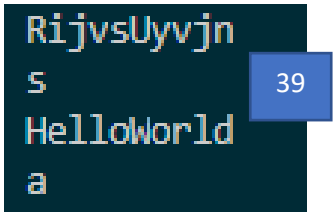
37

```
ADA: Which cipher would you like to see a tutorial of?
You: vig
ADA: Now, onto the tutorial!
ADA: Today we'll be looking at how to decode a Vigenere Cipher if you do not know the key,
but instead a small part of the text. We know "HARRY" is the first in the word in the text,
 but not what the key is, or how long it is. For context, this is a message from two spies
in the world of George Orwell's 1984, so we may be able to guess some key words.
```

38

```
ADA: Which cipher would you like to see a tutorial of?
You: unknown
ADA: Now, onto the tutorial!
ADA: For this tutorial, we know somethings and not others. We have no idea which cipher thi
s is, and therefore no idea on how to decode - yet. By looking at the known parts of the te
xt, we can make inferences and find potential ciphers. The three categories that we will de
al with are: Monoalphabetic, Polyalphabetic and Transposition. The first of these, Monoalph
```

39



Beta Testing

For this stage of testing, I gave a copy of my code to the Code Breaking Club, who used the codes for the Cipher Challenge and attempted to break them using this tool. To fill this table, I used collected feedback from those sessions, however because it was both live and interactive, this is a small summary of the final thoughts after attempting it. Here were the outcomes:

Challenge Number	Cipher	Outcome	Feedback/Observations
1A	Caesar	Solved	Used the Autosolve feature. Extremely useful as solved practically immediately.
1B	Caesar	Solved	They also used the Autosolve on this one. This was really useful!
2A	Affine	Not Solved	The analysis tools were useful here, as it helped identify some letters in the text, however because the affine cipher was not a part of the decoding selection, ADA was a lot less useful here.
2B	Affine	Not Solved	This cipher was also an Affine, reducing the code's usefulness, but the compare worked well when figuring out the affine.
3A	Substitution	Solved	Using some trial and error this was good. Using the SuggestReplacements was key here, and constantly checking back at the original text.
3B	Substitution	Solved	They spotted there might be a key word in this one quite early on, so the suggest replacements was useful to identify that.
4A	Substitution	Solved	The club's intuition is getting much better, as this was solved faster than some before. Unknown whether that was due to ADA or themselves.
4B	Columnar Transposition	Solved	Quite simple to use, using intuition worked out the correct key order
5A	Substitution	Solved	As the substitutions were now an expected thing, that's what they tried first. Once they identified it, the substitution tool was quite effective.
5B	Vigenère	Solved	They noticed that the text was reversed, which started them off well. However, with a little help about the crib, the Vigenère modes were really useful.
6A	Vigenère	Solved	Using a potential crib, the Vigenère was solved relatively easily here.
6B	Substitution & transposition	Solved	This was quite interesting, as when you analysed and used SuggestReplacements, you got some noticeably confident matches on letters. It took a while to realise this was the first of doubled up ciphers.

Challenge Number	Cipher	Outcome	Feedback/Observations
7A	Vigenère	Solved	We didn't have a crib here, however previous experience helped figure out potential information in the text. It was also reversed, which wasn't too obvious at the start.
7B	Vigenère & Transposition	Not perfectly solved	Due to the mysterious layout of this code, they were extremely puzzled by it. One of the members noticed it was in a 47x47 grid, and they found out that it was read in columns, which made it simpler to solve then.
8A	Vigenère	Solved	At first completely bamboozled, but after a few thoughts, the key was figured out. Took them a lot longer here.
8B	Polybius	Solved	Due to the nature of this unknown cipher, mainly the analysis capabilities were used here. They really found the EveryX function useful in this, as it works with bigrams to create a pairing of letter.
9A	Vigenère	Solved	They noticed that it was a Vigenère due to the analysis, however if they thought they got the key they felt they did it wrong. However, the semi-repetitious nature was quickly spotted.
9B	Transposition and Substitution	Not Solved	Due to the fact that this was encoded with Base 24, my project wasn't too useful here, however once we got some help (to tell us that), ADA did manage to help analyse the text well, suggesting replacements for the letters, which aided the transposition.

Evaluation

Completed the Objectives?

One of the main things that shows whether I have achieved what I set out to do, is whether I have managed to complete the goals that I set out for myself at the start of the project. For this, I aim to talk a little about each objective, and explain whether I think that I have Hit, Glanced, or Missed it. If it is a 'Hit', I feel I have completed this objective to the best of my ability, if it is a 'Glanced' that means I feel I have achieved most of the functionality, but there is some room for improvement. 'Missed' is that the objective has not been addressed at all, or hasn't been completed to acceptable functionality, in the project. Just below is a table summarising my thoughts, however I will explain the feedback and my thoughts further below. I will reference and quote feedback, however the full screenshots are in the Appendix. [110]

Objective Number	**Name**	**Hit/Glanced/Miss**
1a	Encrypt a Caesar	Hit
1b	Encrypt a Substitution	Hit
1c	Encrypt a Vigenère	Hit
1d	Encrypt a Rail Fence	Hit
1e	Encrypt a Transposition	Hit
2a	Decrypt a Caesar	Hit
2b	Decrypt a Substitution	Hit
2c	Decrypt a Vigenère	Hit
2d	Decrypt a Rail Fence	Hit
2e	Decrypt a Transposition	Hit
3a	Frequency Analysis	Glanced
3b	English Values	Hit
3c	Index Of Coincidence	Hit
4a	Auto Solve Caesars	Hit
4b	Auto Solve Substitution	Glanced
5a	Tutorial has an intro/outro	Hit
5b	A Tutorial present that walks through set of ciphers	Hit

Reflection on Objectives & Feedback & New Opportunities

Menus

When I first started to receive feedback, it was mainly about the Interface that I had designed, ADA, and how it both helped and hindered at times. One club member wrote that:

> *"I cant [sic] get back to the main menu (once begun encrypting) … [The menu should] always be able to go back to the start"*

This was a really intriguing bit of feedback that I had actually considered when designing the interface and I encountered it in testing, however when I began to design some of the validation, it would require a lot of complexity to then extricate yourself from the loops. However, it can be done, so I will look to implement this into the code in a Version 2, potentially using an interruption style flow. It would be an interesting way to expand the project, to make it more friendly for end users.

I also had some feedback about the menu system from one member of the club saying that they would prefer just a plain design, with some numbered options, rather than the natural-language-style interface. However much I enjoy ADA, I have realised she is a little complex, and I may have over-complicated her over the course of the project.

Someone I also contacted for feedback is James Snape, who has a developer and coding background, however, was really interested in the work I had done in my project. Something which he mentioned in his feedback was:

> *"Menu handles a lot of the validation; this is an interesting design choice"*

I've realised this is because I have 'outsourced' the validation (so to say) to the Menu class, which is because when I was testing the individual cipher functions, I never thought about including the validation into them. This means Menu must do a lot of work, however I won't change the code too much now, as it is slightly larger than expected, however I will definitely plan this and improve it for the future!

The idea from my end user as I finished my design section of including the text input works very well. When I showed it to my users, they had a little difficulty initially, however once they realised it was case-sensitive, all the user-side problems were solved. Overall, it works very well to open the files and I believed I have completed that **objective**, albeit it does not have a specific number like the others. I feel it was useful to include it, as without that, the project may feel incomplete and have a lack of functionality for the club members.

Ciphers

In the cipher classes, they revolve around each of the objectives more than the menu does. The Menu takes all the information, and simply places it into the cipher classes, and then they do the brunt of the work. There wasn't too much feedback from the people I reached out to about the specific encoding or decoding algorithms, but I do have some opinions myself, plus some knowledge from my thorough testing. It may be a good sign that the feedback that related to the cipher-solving algorithms was not too problematic but could be that the flaws of these were encased in ADA's flaws instead.

For the Caesar algorithms, my encode fulfils the **objective (1a)** here. It is able to take all the correct inputs and returns a correct output. My encode also shares a lot of similarities with the decode function, as they share the same base function of solve, and therefore if the encode works, the decode will too (**objective 2a**). You can tell that this empirically works, as during my beta testing

stage the Auto-solve function (**objective 4a**) was able to decode the cipher from the first two challenges (1A/B). I got some interesting feedback from one member of the club, Cali, asking:

"Shouldn't the caesar [sic] decoding function work out the shift itself?"

This situation was partly covered by my acceptable limitations, where I state that my "auto-solve features will only work heuristically, suggesting possible outcomes, and potential matches, but leave it to the user". This is because due to the nature of English, if I ended up implementing something based on simply letter values, it would decode it incorrectly for a smaller piece of text, as the percentages would be swayed a lot more by the repeats of some letters. This would mean every piece of text input would have to be representative of a normal English piece of text, which it is not always. You can see this in my testing with the 1st Chapter of George Orwell's book, 1984. The values for B and W are a lot higher, as a main focus in the chapter is "Big Brother" and the main character is called "Winston". This warps the frequency analysis, and thus the auto-solve. So, whilst this suggestion is a good one, it falls out of scope for this project, however it may be useful for a further iteration of ADA.

An interesting suggestion by James was extending the Caesar algorithm by looking at the ASCII values as I have already included some functionality for it in the project. He suggested:

"Your Caesar cypher is a caesar26 since that's the rotation size. We know there are 128 printable ascii chars though so what if you did a caesar128?"

Because a normal alphabet is 1-26, and ASCII has so many other characters, including it various punctuations, it would involve a completely new way of thinking to try and break it manually. If I look at an ASCII table, the characters 32-126 are all printable (different to James' idea in his feedback) with space being 32. If I shifted all letter by some number between 1-94 (the difference between them) you would end up with something which looks insane but has better security.

From that however, you could also create a new type of Vigenère, where the capitalisation and punctuation in your rolling key would completely affect your encryption or decryption. The Index of Coincidence would also have to be adjusted, as the algorithm only checks for a-z, rather than capitals, lowercase and punctation. This would be a lovely way to expand the program to greater heights, and also provide a new way (for the club members) to see how a simple Caesar can just be manipulated and it becomes a lot more secure.

In my Substitution encryption, a similar problem occurs with its 'auto-solve' feature (**objective 4b**) as does with Caesar's, as it only suggests replacements based upon the percentages of characters in the text. I would change this feature next time, as I would want it to be a little more intelligent, using three letter combos to suggest word replacements too – such as "the" or "and", you could use a frequency analysis and combinations to suggest replacements that way. This means it would be a lot more accurate with longer pieces of text for the cipher challenge. However, it falls into a similar pitfall as the Caesar auto solve does, as a smaller piece of text, such as "Hello World" will fail to be analysed & auto solved correctly due to the fact it is not representative. Usually a cipher challenge piece of text is, albeit, it can end up being completely different, usually in the later ciphers.

Although, on a better note, the encode and decode functions of the substitution (**objectives 1b and 2b respectively**) work exactly as expected. Using the library function, albeit a different way that I imagined I'd complete it when first investigating the project, is an extremely efficient one (it has a time complexity of O(n)). This utilises a map-like function to convert every character in the string to another or specify items to delete. Perhaps if I'd do this again, I will try to recreate this function

myself, as I realise this may have been a shortcut on my part. This would mean I would change my design for the Substitution class entirely. It would be interesting to see how I would have ended up writing the code differently if I had never found this library function in the first place.

Rail fence is an interesting one. When I began this entire project, I felt like Rail Fence ciphers were ridiculously hard, and that never would I get any algorithm to work. However, after carefully going through the algebra, I realised it was simpler than I had expected. The main thing I had problems with at the start was generalising the formula, because I had formulas for 3 rails, or 4 rails, but not for x rails. However, after I got my head sorted on this, the actual encoding (**objective 1d**) worked a lot nicer than expected. I just had to create specific cases for each of the rows, and then it started to all fall into place. My decode algorithm is elegant too (**objective 2d**). Taking ideas from the functional programming implementation that I had been researching, it splits the code into rails, then just takes (pops) the correct letter off each times, whilst maintaining where the program is in terms of the rails. This is also pretty efficient, as you never have to loop over extra characters, just accessing the one you need and moving forward. Overall, I think that my encode function is as best I could get it for this project so far, with my level of experience. On potential expansion of this, I could implement an auto-encoder, as I generate a list of all the potential rail numbers. If I use a random number generator (random in Python) I would be able to select random rail number, and simply just encode it for them. This also solves some feedback from a classmate Lizzy (also doing A-Level Computer Science) who mentioned:

"It would be useful if you could suggest number of rails to decode by"

This occurred because a bug in the code prevents the user from leaving the Rail Fence encoding system without picking a correct number of rails. However, ADA failed to tell the user what rails they can use, unlike with the Transposition. This was a case of accidental mis-implementation, and as soon as I received this feedback, I corrected the error in the code and sent out an updated version to all the members. You can see an updated section of the code in the Appendix. Some other feedback from my Rail Fence section is the idea that is there aren't a suitable number of rails to encode or decode with, it just rejects it. Dr Drape had an interesting other idea:

"If the length of the text means that there are no suitable number of rails then how about adding extra letters (such as X) at the end of the text until, say, 3 or more rails can be used"

This is a wonderful suggestion. It lets the user have a better experience, without being too process heavy. Also, the computation required isn't too complex. Due to the cyclical nature of a rail fence, the number of letters needed to be added is very small, especially for 2, 3, 4 rails. This means you do not have to worry about the while loop that I have used (in my corrected code, see the appendix) never ending, as it is based off a formula that will eventually return a valid output.

This also links to some feedback from my end user, Dr Drape, who said:

"Would it be possible to give a list of suitable rails for the rail fence cipher? Could you encrypt texts randomly?"

He had similar thoughts to Lizzy, as he also received the outdated version of the rail fence menu, however he made an interesting point about the autoencoding. It would be an interesting new feature, that just picked randomly the information required for each cipher. For a Vigenère, for example, the code would have to pick letters a random number of letters, and a random letter between a-z, and then encode it. For Caesar and rail fence, it's quite simplistic. For substitution, it

would be very interesting, as you would have to make sure there aren't any collisions and repetitions of letters. Transposition would have to pick a number of columns, then an order to encode by. This would be lovely extension of the program, and I would love to implement this later in a Version 2 of my code.

The transposition functions are quite efficient too. They use a lot of indexing to collect the correct number from the specific positions, but it uses a lot of loops too. You would expect this to have an inefficient time complexity, however from the way the algorithm works, each letter should only be accessed once, as it skips over the incorrect letters. This, in my opinion, works well for the solution I am offering. If I had decided to do the different version of the Transposition (using rows instead of columns), then all my algorithms would completely change, however it wouldn't affect the efficiency as a similar amount of operations will be occurring. My Transposition encode (**Objective 1e**) works by taking each letter from that section, but in the order of the key specified by the user. My decode works similar (**Objective 2e**), however depending on the input of the user, it will potentially calculate the reverse of the key. This is a really interesting algorithm as it completely swaps the text in an unexpected way, basically unpicking the encryption.

My Vigenère algorithms are quite similar to my Caesar ones, except the main encode (or decode) functions deal with a small amount of processing. They make sure the key loops, then simply puts the information into a function that does the calculation. However, when I am doing this, I use the ascii values for an uppercase or lowercase letter a lot. As James pointed out:

> *"Magic numbers are what we call constants in code that require extra knowledge to understand. If you know what the value means brilliant but if you don't its like not being in on the joke…. When you look through your code you will see a lot of constant values that aren't stored specifically."*

You do see this with Vigenère and in my Caesar, where I have the constants of the ASCII values, and for the Caesar, 26 because it is the length of the alphabet. In order to prevent these "Magic numbers", then I would instead place them as variables inside on either the Cipher class (if they are across 2 ciphers) or just a variable in the class normally (a self.__VARIABLE in Python). It would also improve the readability of my code, especially if I were to hand this to another developer.

Other than that improvement however, I believe my Vigenère algorithms themselves are okay. My encode function (**objective 1c**) can work for both uppercase and lowercase letters (see the testing section), and there is sufficient validation in the Menu functions to prevent any incorrect encodings. This algorithm is also sufficiently fast, as it can cope with much larger quantities of data than the club members would usually work with, which is definitely a positive here. The decoding ability (**objective 2c**) has similar properties to the Encoding, as they are based on the same algorithm, with the same protection and validation. When I discussed the algorithm with my end user, he thought the current solution worked well but said:

> *"You could use [an] XOR (meaning the same key can be used for encryption and decryption)"*

This is such a cool idea. This would make the Vigenère act like the Vernam Cipher (on the A-Level Computer Science syllabus) and assist in explaining the cipher to a computer science student and demonstrating the encryption. However, I must add it will not be a Vernam, as it requires a key longer than the length, whilst a Vigenère uses a rolling key. Something else which I would improve the solution would be an automatic Vigenère decryption algorithm, or at least a best-guess scenario decryption. This would use a slightly modified version of the Index of Coincidence, which can give

you an indication of the key length, and using a crib of the user's entering, find a repeating piece of text

However, one of the most useful aspects of ADA to the club members, is her tutorial section (**Objective 5a and 5b**). ADA begins by introducing each of the ciphers (**Objective 5a**), but then for the rest of the tutorial, takes the next item off the queue, and prints out the two pieces of information until it reaches the last element, which are the explanation for the tutorial. Before deciding on the tutorial explanations, I made sure to ask people who have no cipher-solving experience, and people who have solved ciphers before, to make sure all the explanations were of adequate simplicity whilst not being over-simplified. In the areas of the code that I had not achieved a programmable solution for, such as a Vigenère that can be solved with a crib to work out the rolling key. Whilst you can simply attempt to decode using the suspected crib, it is not as smooth a process as I would like, hence why I included a tutorial on it. Same with the Unknown cipher, it walks a beginner through the process of identifying the cipher from the frequency analysis and the Index of Coincidence, which satisfies partly **Objective 3)a)iv),** however for full explanation see later on. However, some of the feedback from Dr Drape said:

> *"I think it would be helpful to have more pauses and put the information in smaller chunks. Maybe ask the user to press a key to move on (which also gives the user the chance to quit if wanted)."*

I understand where this feedback is coming from. There is a lot of relevant information for some of the ciphers, and especially with some of the more complex decryption tutorials, the user may get lost in the information, even with the inbuilt pauses. The idea I like out of this is the user input to carry on, as it lets the user pace the code, and allows it to cater for different speeds of readers. I will definitely change some of the tutorial to include the user input or ask them to input how long they would like the tutorial to pause for. I will update this in a V2 of ADA however, as it will require a mild re-design of the tutorial.

Analysis

The analysis part of the code is completely different to the cipher area. This is instead focusing on the specific of changing the text, now the program is focusing on number crunching and more process heavy algorithms.

The Frequency Analysis capabilities in ADA are very good (**Objective 3a**). You have the option to see the normal frequencies of each letter, in the Menu option Frequency Analysis, and the Compare option shows the percentages of each letter compared to the percentages you would expected from a normal English piece of text (**Objective 3b**). This all works great; however, it has a small amount of latency (in the order of seconds) when using extremely large pieces of text, however for the Cryptography Club it works sufficiently quickly.

However, I do not believe that I have achieved the **Objective 3a** perfectly. This is because whilst I inform the user that: an Index Of Coincidence close to 1 is likely a Vigenère; that the Frequency Analysis having similar percentages to English but across different letters is likely a substitution; that having nearly the same percentage across similar letters is likely a Rail-Fence or Transposition, I never actually have a piece of code which informs them this. This misses the **Objective 3)a)iv)** from my analysis, where I aim to inform the user of this. I think I didn't end up achieving my objective because this objective is based on a lot of contextual knowledge, and that in reality you need a human's intuition to fully understand whether it is one or another. You may also encounter ciphers

that are multiple (as we saw towards the end of the cipher challenge). So, whilst I am upset that I haven't hit all my objectives perfectly, this was an objective that looking back, wasn't the best.

Another key piece of **Objective 3a**, is the most common sets of letters. I have partly changed the idea, rather than the most common sets of 3,4,5 letters, the user can decide what they would like to be combined. This fits perfectly the solution without causing much latency. However, something which I would change for the future would definitely be the way that we take these combinations. Currently I am using a "sliding" method, which basically includes each letter multiple times. However this method is a bad idea for some ciphers, like the Polybius which we encountered in the Cipher Challenge this year, which uses a pair of letters (a bigram) in place of a letter, as it has a 5x5 grid with coordinates designated as letters. If I was trying to do a bigram with my sliding algorithm however, there would be letter counted twice, which is not a suitable way of deciphering or analysing this cipher. I would next time incorporate two methods of bigrams, one which is "static".

Another key piece of the Analysis section is the Index of Coincidence (**Objective 3c**). This is the value that lets us know how random the text is and is mainly relevant to the identification of the Vigenère cipher. However, there isn't too much you can do with it. When I was talking to my End User, Dr Drape, we discussed how I could potentially make the Index of Coincidence information more useful:

> *"Index of Coincidence - what about giving a measure (maybe a percentage) to show how close the ciphertext is to normal English text."*

This would give a physical metric to the Index of Coincidence value, as to a user they may only recognise it as being "close" but not *how* close. A simple calculation of the percentage difference would really help visualise clearer how close to English the text is. This links to some feedback from James, who suggested:

> *"You know what the properties of the encoded result should be - uniform letter and word distribution, random etc you could write a function that scores the output of the cipher i.e. your cipher score is how good your result is."*

This is a really clever idea, as it can let the user known how "encrypted" their cipher is. It isn't too useful in the sense to aid with the code-breaking competition; however, it can also be used if you are sending encrypted messages. This would be an interesting area to expand into, as I have not thought about this area until now. From this, you could expand into other ciphers, and if you got a full set of auto-decoders working, you could measure how long it takes to decrypt the code, to how well encrypted it is. This would work great in showing time complexity and how computable some ciphers are, especially because in our A-Level we cover ciphers and their computational security.

Next Steps

Now that I have collected and responded to all the feedback, I aim to correct some faults in the code, as some small fixes were suggested. This will then be re-released to the members in order to assist them when the competition is begun again. I don't believe I will release any of the Improvements or Expansions at that point in time however, as they will require more thought process than some changes of menu structure for example.

Some of the minor changes to be made (as mentioned above in Reflection on Objectives & Feedback & New Opportunities) are: Changing the ascii values used in Caesar and Vigenère to be constants; an option to reverse the text in the analysis section, an ability

However, something that I would love to explore between then and now with my code, is how "correct" it is. During the testing of my project I came across pylint[12] to help find simple bugs or mistakes, and check if my way of writing it is "idiomatic", or accessible for other Python developers to understand my code. This would be an interesting step forward to look at the way the code should be for an actual project, rather than just the way I have been writing it.

From this idea of using Pylint, you can use it to analyse the code as I currently have it. The finished code so far (with all of the edits) has a score of 9.18/10. This is very good, as 10 is the best code ever, however this is calculated using certain available modules that you can turn on or off. In my case, I have disabled the format, basic and design functions. This is because of the idiomatic style (which I explained earlier) is not present in my code, so 'basic' has been disabled as it is not relvenat to this project. Similar to that, the format simply detects lines which are too long, and design tells you whether you have too many functions (or too little). In a perfect world, I would not disable all of these, but due to the nature of this project, they all fall out of scope for this version of ADA. To see the exact call I use in the Windows command line, please see the Appendix.

Other, more advanced, potential areas for expansion are involving a greater aspect of the natural language. In this, I mean implementing a truly natural language menu system, where it can deal better with human inputs. From this, it would be useful to actually implement an AI or machine learning algorithm that does learn from the user and their inputs. It would be great to also use a similar machine learning algorithm, but instead apply it to the texts, and simply tell the AI what information you know (that it is a Caesar, or a Vigenère has a key of length 4), and it may be able to use its previous experience to use this to automatically decode this. I do not have the current knowledge or expertise in such an area, hence why it was not possible in this specific project, however there is nothing stopping it.

Because of this, it may be useful to store this computing power, not on the device you are using but elsewhere. You could implement a Cloud capability, potentially using Microsoft Azure, which would reduce the load on the laptop, and also provide the same service to multiple people, whilst learning and storing the data in one place. This would be really useful especially if you didn't have that, the machine learning algorithm would be restarted every time the code would close, and different users would end up with a different cipher algorithm, which wouldn't be sensible. This is also remarkably close to the idea of an API at that point. This also means that you can call a webservice, and it will return to you a relevant output. You can already see this style of design in the way my Cipher classes are set up. However, in this situation, the error checking and validation would have to be moved into the specific cipher classes. I mentioned this earlier, and definitely would like to re-design the classes to incorporate this.

In my current menu system, the user is allowed to input text or open a file to use. I specify in my acceptable limitations that a normal text file is all that it can open, it doesn't accept a .bin or a .pdf. In the future, it would be interesting to also be able to open or read these files. I would need to do some research into the make-up and composition of these files. Another form which could be useful is a .json file which could be used to link in with the API idea, as I know that json is used to transmit data when talking to certain APIs.

Another interesting improvement would be to create a "profile" type feature, which allows the program to store some small pieces of information about the user, such as a name and what stuff they use the most. This would be used to change how options are presented to the user in the menu

[12] https://www.pylint.org/

system, and if this project was moved online (or to a webpage), I could return anonymised data to then help improve the project further. However, this is extremely complex at the moment, and some of these points fall under the GDPR, which I would need to fully understand before heading along this design path. But if I had a lot more experience, this would be a great idea, tailoring the experience to the user's past, and hopefully making their experience a lot smoother.

However, after all of this, the simplest idea to extend it would be to use a GUI using Tkinter in Python. This will allow more interactivity in the code, and will drastically reduce the potential problems with the user input and validation, as you can completely lock the user out of weird input if you are using certain types of inputs (like rolling box or radio button). This would be an appealing new function that would not affect the current implementation too drastically.

Appendix

Survey

Cryptography Project

1. Which of these would interest you most?

- ◯ A large number of ciphers to decrypt (but manually)
- ◯ A really effective automatic solver, but choose which type of cipher
- ◯ An automatic solver that's slower but you don't need to choose the cipher type
- ◯ Other (please specify)

2. What Ciphers would you most want to be able to decrypt?

3. Would you like to see an interactive tutorial on how to break a select few ciphers?

- ◯ Yes
- ◯ No

4. Do you care about a long latency? (latency means how long the program takes to output an answer)

- ◯ The most important priority
- ◯ A top priority, but not the most important
- ◯ Not very important
- ◯ Not important at all

5. Would you want an snazzy interface or better cipher solving capabilities?

- ◯ Interface
- ◯ Better cipher solving
- ◯ I don't really care

Feedback Screenshots

RE: Exciting Program and Feedback!

Carolina Stott

Sent: 07 March 2020 19:09

To: Lucy Jefford

shouldn't the caesar decoding function work out the shift itself? or is that quite hard to do

---

RE: Exciting Program and Feedback!

Carolina Stott

Sent: 07 March 2020 19:10

To: Lucy Jefford

also once im in the caesar bit i cant get back to the main menu so i think you should always have an option to go back to the start

---

RE: Exciting Program and Feedback!

Carolina Stott

Sent: 07 March 2020 19:17

To: Lucy Jefford

i think for the caesar shift (just an idea) for the autosolve option you could find a way to work out which sentences are definitely not options (if the user has specified the sentence is in english) maybe by looking at letters that never appear together in english (eg q always has a u beside it) im sure theres a list online, or also some letter are never by themselves like h and stuff and then it could refine the options for the sentences. might be quite a stupid idea as its probably quite hard to code something like that but maybe u can talk about it in the log

---

RE: Exciting Program and Feedback!

Matilda Knight

Sent: 08 March 2020 11:47

To: Lucy Jefford

Hi Lucy,

I've been looking through your program and I really enjoyed using it - the tutorials were really helpful and explained the different types of ciphers really well to a beginner. I tried out all the different ciphers and they all worked really well and quickly. It was all very self explanatory to use and a really cool program.

The only thing that I thought could be different was in the Substitution cipher, when I entered in the same replacement for a couple, it was quite long to have to enter in every letter of the alphabet again - you may have already considered this and it could be really hard to code so I totally understand, but would it be possible to only have to reenter the values which were entered in as the same?

Thanks, Tilda

RE: ADA - code

Stephen Drape

Sent: 08 March 2020 14:00

To: Lucy Jefford

I've only had a very brief play around with this - perhaps you can guide me through it a bit on Tuesday (or in our tutor meeting).

- What's your favourite thing about this project, or what do find the most interesting?

I like the interactive nature - seems to be fairly helpful and intuitive

- What is your worst thing about the code? Do you have any absolute criticisms or points of failures? (where the code is now non-functional and doesn't meet the requirements)

Sometimes the options are not clear. Also the tutorial spits out quite a lot of explanation. Maybe ask the user to press a key to continue after each block of text.

- Do you think this is accessible for the level of the club?

Yes I think so

- Explain what you think could be better for the users, advanced or beginners.

Not sure at this stage

- What are your thoughts on the design of the User Interface and ADA herself?

A bit more guidance in places. For example, would it be possible to give a list of suitable rails for the railfence cipher? Could you encrypt texts randomly? (This might be a feature that I've not got to yet).

- Do you have any areas which could use improvement, or that you find minor problems with?

I couldn't get it to recognise the text file - maybe it's my system.

- In the beginning of the project, you mentioned that the code had to be 'adequately fast'. Does this still meet that requirement?

Seems to be quite fast.

- Is there any feedback which does not fall under any of these questions?

Not at the moment.

NEA comments

Stephen Drape

Sent: 12 March 2020 18:28
To: Lucy Jefford

Hi Lucy,

As discussed, here are some comments about your NEA:

- Tutorial - there is quite a lot of information, I think it would be helpful to have more pauses and put the information in smaller chunks. Maybe ask the user to press a key to move on (which also gives the user the chance to quit if wanted).
- Some colour might be useful to aid readability - although I know this is difficult to do via IDLE so might not be feasible.
- Railfence: if the length of the text means that there are no suitable number of rails then how about adding extra letters (such as X) at the end of the text until, say, 3 or more rails can be used
- Substitution - how about allowing the user to enter a string which represents the encryption alphabet or using a key word to generate an alphabet
- Vigenere - you could use XOR (meaning the same key can be used for encryption and decryption) or extending the encryption to use all printable ASCII characters
- Index of Coincidence - what about giving a measure (maybe a percentage) to show how close the ciphertext is to normal English text.

Hope this is useful

Best wishes

Dr Drape

Re: Exciting Program and Feedback!

Elisabeth Herratt

Sent: 10 March 2020 12:13
To: Lucy Jefford

Hi Lucy,

I've got you some feedback:

For the transposition cipher:
ADA: You didn't include all the column positions in your answer... Please try again!
ADA: Position 0 will be?
You: 0
- this gave an error message. I just input 0 and it said index was out of range around line 254 maybe? I've closed the window now so can't remember exactly what I tried to encode. Sorry!

From a user experience point of view, I would prefer a menu option where you can just enter 1,2,3 or 4 instead of the name of the cipher I want to work with.

For rail fence encoding, it would be useful if you could suggest number of rails to decode by. Coz I don't know what number to choose.

You may not want to do this, but I think it would be cool if you showed the steps that the program takes to decode/encode each cipher. E.g. "shifting each letter by 6", or showing the rails for the rail fence?

I liked how there was a time delay between responses, it gave me time to read what ADA was saying. The time taken for encoding and decoding was really fast which I appreciated. I also enjoyed the tutorial, the walkthrough was very clear and concise. Congrats on a great project!

Hope this is helpful,
Lizzy

Re: Exciting Program and Feedback!

Amena Boyd

Sent: 10 March 2020 19:04
To: Lucy Jefford

hey it looks good - only thing is that it would be useful if you indicate what a valid input is (e.g. yes/no) in the question so I won't put Y assuming it's valid and get sent back to the menu. Also I found it kinda hard to get out of the 'layers' of menu choice - e.g. if i was working in caesar and wanted to switch to vigenere, it was kinda tricky to figure out how to do that at first. Overall, really cool though!! I really liked the tutorial bit

Pylint reference

pylint --disable=format,basic,design "C:\Users\LucyJ\OneDrive\School\A Levels\Computer Science\NEA\Actual\Cryptography.py"

Code Updates

RailFence (Suggesting Rails When Encoding/Decoding)

```
def RF_EncodeDecode(self, encode=True):
        if encode:
            print("ADA: The Rail Fence cipher is complicated to encode, but it
's simple with a couple of key pieces of information.")
        else:
            print("ADA: The Rail Fence cipher is complicated to decode, but it
's simple with a couple of key pieces of information.")
        print("ADA: One of these is the number of rails, the other is your tex
t! ")
        ##Get text
        text = self.GetText(remove=True)
        railCorrect = False
        while railCorrect == False:
            try:
                ##Get rail number (2 or higher)
                if encode:
                    railNum = int(input("ADA: How many rails do you want to us
e to encode this with? \nYou: "))
                else:
                    railNum = int(input("ADA: How many rails do you want to us
e to decode this with? \nYou: "))
                ##If rail number in the array returned by GuessRails
                rfRails = RailFence(ciphertext=text, railNo=railNum)
                guess = len(text) #Check all potential rails in case it doesn'
t exist then
                rfPotentials = rfRails.GuessRails(guess)
                if railNum in rfPotentials:
                    railCorrect = True
                elif rfPotentials == []:
                    print("ADA: I'm so sorry but the text you are trying to en
code has no suitable rail numbers to encode with. I'm going to have to ask for
 your text again.")
                    text = self.GetText(remove=True)
                else:
                    if encode:

                        print("ADA: In order to properly encode the Rail Fence
 cipher, you need a rail number that, when you do the modulus of the length of
 text by ((2*rails) -
2), it should be 1. \nADA: Please try a different number, as this wasn't accep
table.")
```

```
                        print("ADA: How about using ", str(rfPotentials), "to
encode your text with?")
                    else:
                        print("ADA: In order to properly decode the Rail Fence
 cipher, you need a rail number that, when you do the modulus of the length of
 text by ((2*rails) -
2), it should be 1. \nADA: Please try a different number, as this wasn't accep
table.")
                        print("ADA: How about using ", str(rfPotentials), "to
decode your text with?")
                except:
                    print("ADA: That doesn't seem right. Could you try something d
ifferent perhaps?")
            if encode:
                print("ADA: Now I'm going to encode it for you.")
                railEnc = RailFence(railNo=railNum, ciphertext=text)
                railEncoded = railEnc.encode()
                print("ADA: Here is your encoded Rail Fence:", railEncoded)
            else:
                print("ADA: Now I'm going to decode it for you.")
                railDec = RailFence(railNo=railNum, ciphertext=text)
                railDecoded = railDec.decode()
                print("ADA: Here is your decoded Rail Fence:", railDecoded)
            time.sleep(3)
            print("ADA: Heading back to the Rail Fence menu now!")
            time.sleep(1.5)
```

RailFence Extra Characters

```
elif rfPotentials == []:
                    if encode:
                        print("ADA: I'm so sorry but the text you are trying t
o encode has no suitable rail numbers to encode with. I'm going to have to ask
 for your text again.")
                    else:
                        print("ADA: I'm so sorry but the text you are trying t
o encode has no suitable rail numbers to decode with. I'm going to have to ask
 for your text again.")

                    #### Loops through this until RFPotentials actaully has so
mething that works. Lets the user know how many extra characters they had to a
dd
                    addChoice = input("ADA: Would you like me to instead add o
n characters ('X') to your text (you have just picked) until there is at least
 one valid number of rails? \nYou: ")
                    if self.PositiveResponse(addChoice):
                        correctRails = False
                        counter = 0
                        while correctRails == False:
                            counter +=1
```

```
                            text += "X"
                            rfRails = RailFence(ciphertext=text, railNo=railNu
m)
                            rfPotentials = rfRails.GuessRails(len(text))
                            if railNum in rfPotentials:
                                correctRails = True
                        if encode:
                            print("ADA: I have added", counter, "letters to yo
ur text. You can now encode it with", rfPotentials)
                        else:
                            print("ADA: I have added", counter, "letters to yo
ur text. You can now decode it with", rfPotentials)
                    else:
                        print("ADA: Okay, then you'll have to enter in a diffe
rent piece of text, as that wasn't valid.")
                        text = self.GetText(remove=True)
```