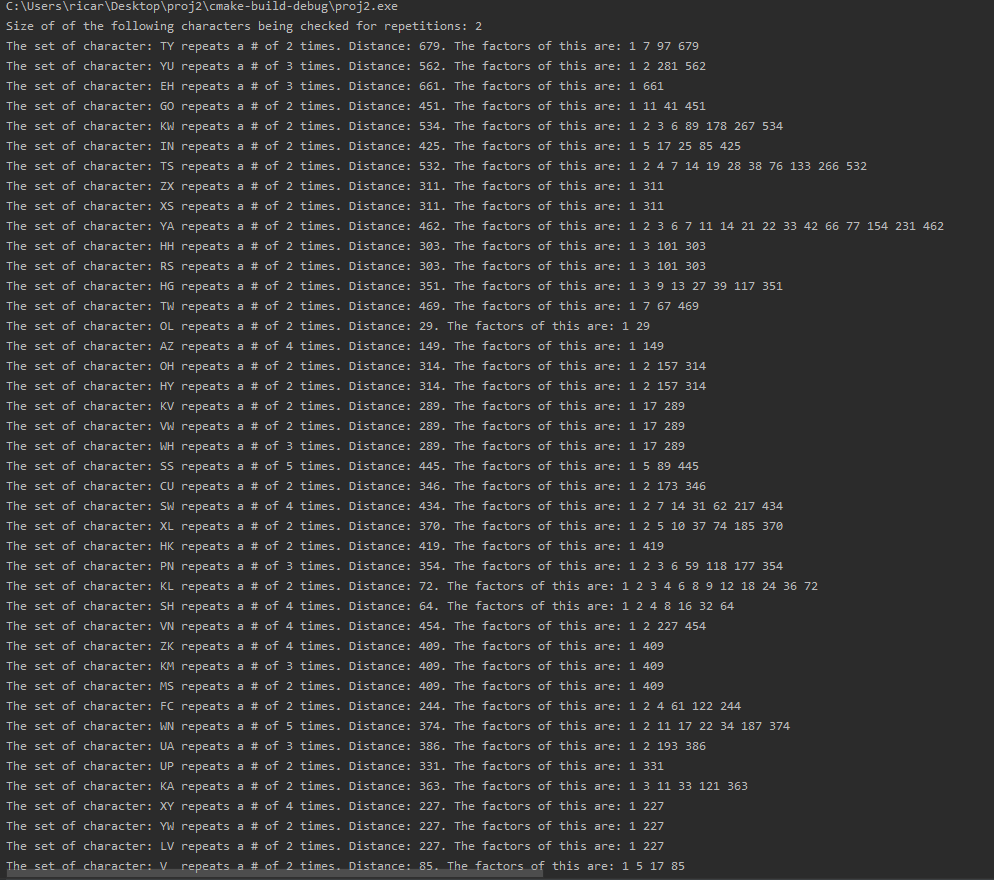
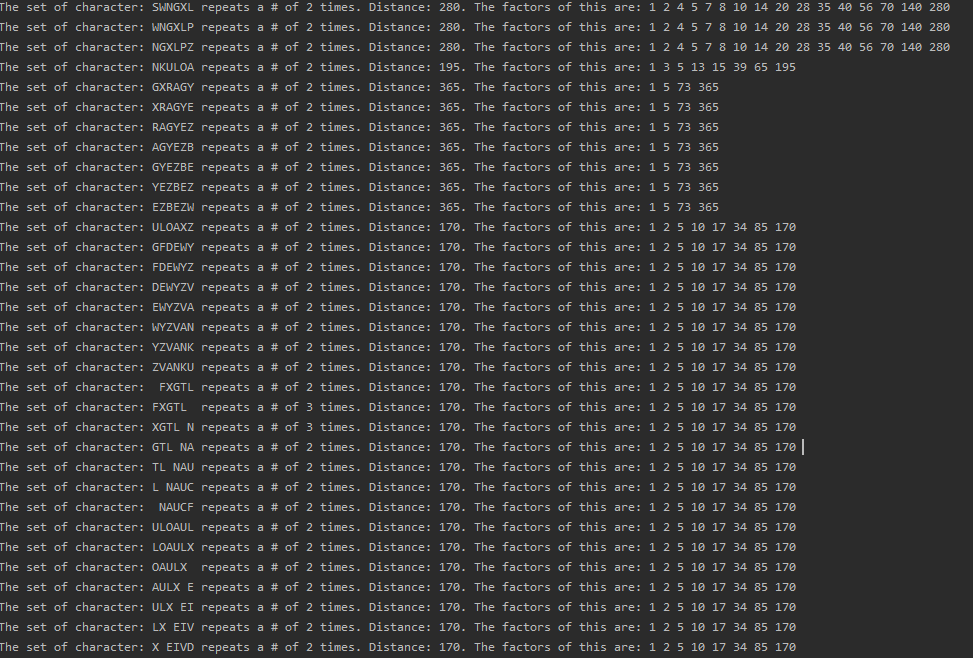
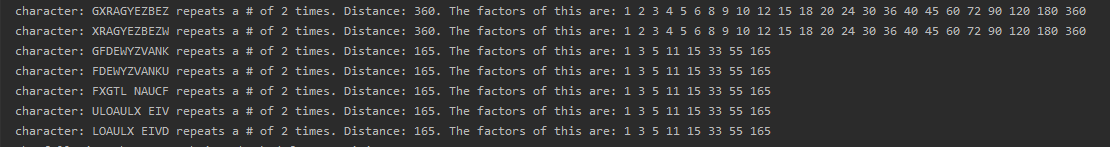
Step 1:

My program goes through all the characters so it might search for a set of characters that have been already found. Therefore, I just attached a couple of pictures of it. In addition, I added a text file called “Step1.txt” where it shows every single repeating set of characters with their distance and factorization.





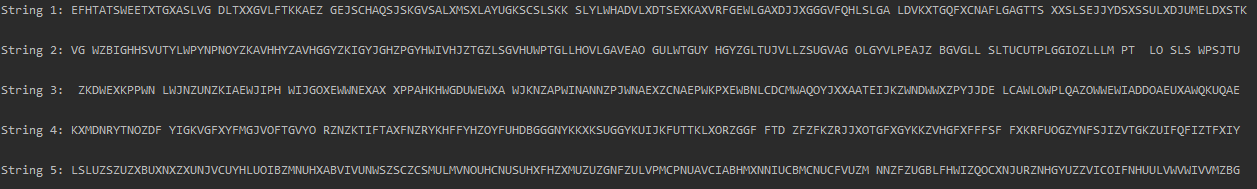


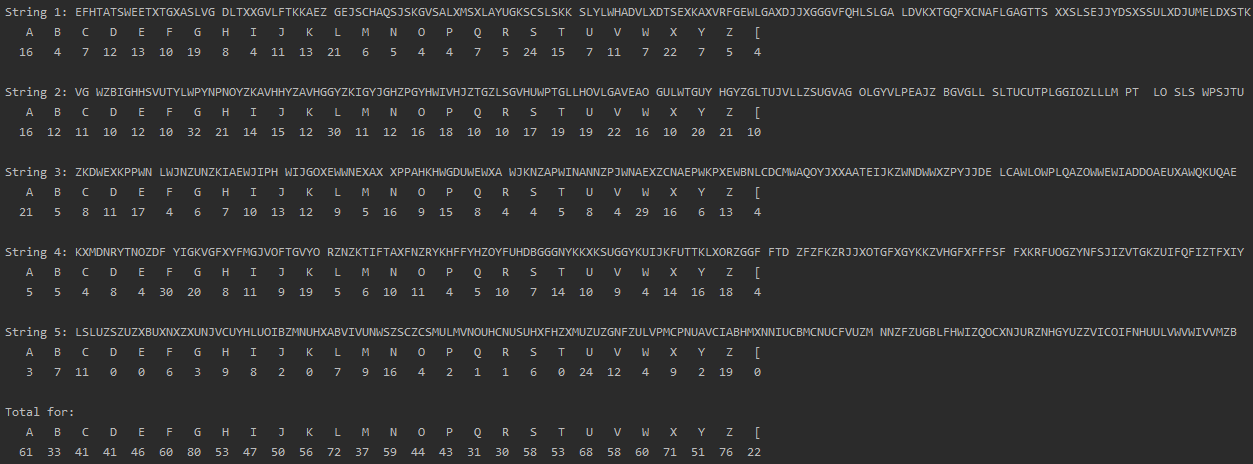
Step 2:

While there were many factors that keep showing a lot. I for sure did not believe that the key was going to have a length of 1. I also knew that the key was at most 10 characters long, based on the instructions. I focused mostly on larger repetition patterns as saw on the screenshots for Step 1. I quickly noticed that the factor of 5 kept showing a lot. Characters sets with a length of five repeated a lot. But most importantly, the largest character repetitions with the length of eleven character had the factor of five. Therefore, I decided to use 5 as the length of the key.

After, having issues trying to create the new strings grabbing every fifth character I decided to use python, since there were much more resources that showed you how to do it.

The following picture shows the strings being shifted by five.

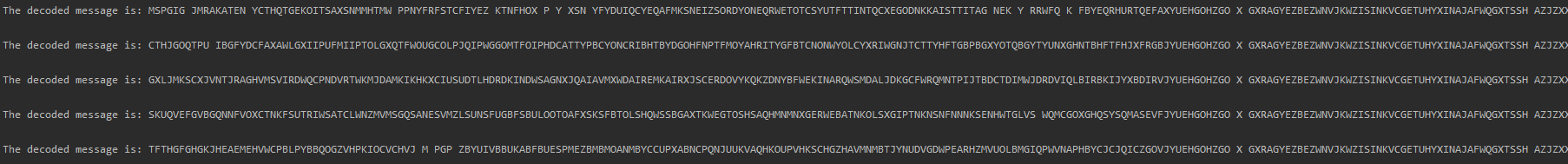




After running the 1-gram model my first reaction after comparing it with number in the Frequency Chart was that G , X and Z where heavily used in the cypher but in the frequency table they are one of the least used ones. The letter E should be the most used one, but it is not. So, I chose G since it is the most used one as my first letter for my key. Then I used X with the same logic behind.

Step 3:

For Step 3, since at this point, I was using two different programs, one in python that shifts my string into the size of my guess key size, and the remaining was done in my C++ program. Most of the stuff ended up getting hard coded with uncountable trials of copying and pasting. After, taking my guess with G and X. I had to decode every single string. Then I got the following output like this:



Then, I would have to copy those strings manually into my python program, which would then realign it. I would then try to see if I can make up words. If not, then I would manually have to copy that string and run it again into the part of my C++ program that would do the 1-gram character analysis. I would slowly start to see that the frequency was getting closer and closer to the real one. I tried thousands of combinations



Step 4:



In the end after, getting help from some classmates I managed to find the combination, which is THXGV.

Step 5:

The following part is my C++ code.

#include <iostream>  
#include <string>  
#include <string.h>  
#include <algorithm>  
#include <iomanip>  
  
using namespace std;  
// Function that counts how many times each characters repeats in the cypher ( 1 gram analysis)  
void countChar(string str, double numRepeats[], double sizeCypher, double totalRepeats[]) {  
 double count;  
 char x = 'A';  
 for(int j = 0; j <= 26; j++) {  
 count = 0;  
 for (int i = 0; i < str.size(); i++){  
 if (str[i] == x) {  
 count++;  
 }  
 else if( str[i] == ' '){  
 count++;  
 }  
 }  
 if(count !=0){  
 // numRepeats[j] = (count/str.size());  
 numRepeats[j] = count - 1;  
 totalRepeats[j] += count - 1;  
// cout << "The letter: " << x << " repeats a total of " << count-1 << " times." << endl;  
 cout << setw(8) << x << setw(8);  
  
  
 }  
 x++;  
 }  
 cout << endl;  
 for(int j = 0; j <= 26; j++) {  
 cout << setw(8) << setprecision(4) << numRepeats[j]/str.size() << setw(8);  
  
 }  
}  
  
  
/// This decodes the cypher  
void decode(string cypher, string temp, string key, int keyIndex, int keySize, string decoded){  
 int hold;  
 char chold;  
 temp = cypher;  
 replace(temp.begin(), temp.end(), ' ', '['); // Replaces every space character to [ (It is the following character in the ASCII table after Z. (makes it easier for me to shift into the proper character while being decoded)  
 for(int i=0; i < cypher.length(); i++){  
 hold = int(temp[i]-key[keyIndex]); // This changes does the operation to subtract the characters and returns the value in integers (So that I can later use operations with the actual integer value)  
 if(hold < 0){  
 hold = (hold%26+26)%26; // This makes it so that if the result of the subtraction of the original character - the key character is lower than A, then it will loop back to space character. Ex. 'A' = 0. 'C' = 4. 'A' - 'C' = -4 which = 'X'  
 chold = 'A';  
 chold = chold + hold + 1;  
 decoded[i] = chold;  
 keyIndex++;  
 if(keyIndex > keySize){ // If the key reaches it last index, it resets it back to the first one  
 keyIndex = 0;  
 }  
 continue;  
 }  
 hold = (hold)%26;  
 chold = 'A' + hold;  
 decoded[i] = chold;  
 keyIndex++;  
 if(keyIndex > keySize){ // If the key reaches it last index, it resets it back to the first one  
 keyIndex = 0;  
 }  
 }  
 replace(decoded.begin(), decoded.end(), '[', ' '); // After the string is decoded, it now changes all the characters [ into spaces ( the proper character)  
 cout <<"The decoded message is: " << decoded << endl;  
 cout << endl;  
}  
  
  
int main() {  
 string cypher = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXYCUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK";  
 int sizeCypher = cypher.size(); // Variable that holds the size of the cypher we know that is 836.  
 string decoded = cypher; // String that by the end will hold the plaintext of the cypher  
  
  
 // After running the python program , we go the following strings. This strings will be used to find their 1 gram character analysis  
 string str1 = "EFHTATSWEETXTGXASLVG DLTXXGVLFTKKAEZ GEJSCHAQSJSKGVSALXMSXLAYUGKSCSLSKK SLYLWHADVLXDTSEXKAXVRFGEWLGAXDJJXGGGVFQHLSLGA LDVKXTGQFXCNAFLGAGTTS XXSLSEJJYDSXSSULXDJUMELDXSTK";  
 string str2 = "VG WZBIGHHSVUTYLWPYNPNOYZKAVHHYZAVHGGYZKIGYJGHZPGYHWIVHJZTGZLSGVHUWPTGLLHOVLGAVEAO GULWTGUY HGYZGLTUJVLLZSUGVAG OLGYVLPEAJZ BGVGLL SLTUCUTPLGGIOZLLLM PT LO SLS WPSJTU";  
 string str3 = "ZKDWEXKPPWN LWJNZUNZKIAEWJIPH WIJGOXEWWNEXAX XPPAHKHWGDUWEWXA WJKNZAPWINANNZPJWNAEXZCNAEPWKPXEWBNLCDCMWAQOYJXXAATEIJKZWNDWWXZPYJJDE LCAWLOWPLQAZOWWEWIADDOAEUXAWQKUQAE";  
 string str4 = "KXMDNRYTNOZDF YIGKVGFXYFMGJVOFTGVYO RZNZKTIFTAXFNZRYKHFFYHZOYFUHDBGGGNYKKXKSUGGYKUIJKFUTTKLXORZGGF FTD ZFZFKZRJJXOTGFXGYKKZVHGFXFFFSF FXKRFUOGZYNFSJIZVTGKZUIFQFIZTFXIY";  
 string str5 = "LSLUZSZUZXBUXNXZXUNJVCUYHLUOIBZMNUHXABVIVUNWSZSCZCSMULMVNOUHCNUSUHXFHZXMUZUZGNFZULVPMCPNUAVCIABHMXNNIUCBMCNUCFVUZM NNZFZUGBLFHWIZQOCXNJURZNHGYUZZVICOIFNHUULVWVWIVVMZB";  
  
  
  
 string temp = cypher; // Variable that holds the the cypher text for when we need to modify/use the cypher text.  
  
 int keyIndex = 0; // Key Index for when iterating through the key  
 int keySize = 4; // Size of the Key /// WE KNOW THAT THE LENGTH OF THE KEY IS AT MOST 10.  
 string key = "THXGV";  
  
 int tempNum = 0;  
 double answer[27] = {0};  
 double characterFrequency[27] = { 0.080, 0.015, 0.030, 0.040, 0.130, 0.020, 0.015,  
 0.060, 0.065, 0.005, 0.005, 0.035, 0.030, 0.070, 0.080,  
 0.020, 0.002, 0.065, 0.060, 0.090, 0.030, 0.010,  
 0.015, 0.005, 0.020, 0.002, 0.018 };  
  
 double numRepeats[27] = {0}; // array 0 = A, array 26 = space. This variable has stored the frequency that each character appears in the cypher.  
 double totalRepeats[27] = {0};  
  
 /// THIS FINDS REPEATING SET OF CHARACTERS, DISTANCE, AND FACTOR.  
 string CharCombTarget[1672] = {};  
 string CharComb2[1672] = {};  
 int position = 0;  
 int numOfChar = 2;  
 int occurences = 0;  
 int secondPos = 1;  
 int distance = 0;  
 for(int k=0; k < 25; k++){  
// cout << "Size of of the following characters being checked for repetitions: " << numOfChar << endl;  
 for(int i=0; i < sizeCypher; i++) {  
 CharCombTarget[i] = cypher.substr(position, numOfChar);  
  
 for(int j=0; j < sizeCypher; j++) {  
 CharComb2[j] = cypher.substr(secondPos, numOfChar);  
 if(CharCombTarget[i] == CharComb2[j]){  
 occurences++;  
 if(secondPos - position > 0){  
 if( occurences = 1) {  
 distance = ((secondPos+1) - position) - numOfChar;  
 }  
 }  
 }  
 secondPos++;  
 }  
  
 if(occurences > 1 && numOfChar > 4){  
// cout << "The set of character: " << CharCombTarget[i] << " repeats a # of " << occurences << " times. " << "Distance: " << distance << ". The factors of this are: ";  
// for(int p = 1; p <= distance; ++p)  
// {  
// if(distance % p == 0)  
// cout << p << " ";  
// }  
// cout << endl;  
 }  
 secondPos = 1;  
 occurences = 0;  
 position++;  
 if(numOfChar+position >= sizeCypher){  
 break;  
 }  
 }  
  
 numOfChar++;  
 position = 0;  
 }  
 cout << endl;  
  
  
  
 /// Counts how many times each character repeats in each string found with the paython program (:  
 cout << "String 1: " << str1 << endl;  
 countChar(str1, numRepeats, str1.size(), totalRepeats);  
 cout << endl << endl;  
 cout << "String 2: " << str2 << endl;  
 countChar(str2, numRepeats, str2.size(), totalRepeats);  
 cout << endl << endl;  
 cout << "String 3: " << str3 << endl;  
 countChar(str3, numRepeats, str3.size(), totalRepeats);  
 cout << endl << endl;  
 cout << "String 4: " << str4 << endl;  
 countChar(str4, numRepeats, str4.size(), totalRepeats);  
 cout << endl << endl;  
 cout << "String 5: " << str5 << endl;  
 countChar(str5, numRepeats, str5.size(), totalRepeats);  
 cout << endl << endl;  
  
 char x = 'A';  
 cout << "Total for: " << endl;  
 for(int j = 0; j <= 26; j++) {  
 cout << setw(4) << x << setw(4);  
 x++;  
 }  
 cout << endl;  
 for(int j = 0; j <= 26; j++) {  
 cout << setw(4) << totalRepeats[j] << setw(4);  
  
 }  
 cout << endl << endl;  
  
 keySize = 3;  
 keyIndex = 0;  
 key = "TXV";  
  
 decode(cypher, temp, key, keyIndex, keySize, decoded);  
 string txv ="MZFXTJMMEWNNSQRGDHJHICBAGXHRZWYVEBEJOOZHMLVAGINCWAZFDCHKCRJHAYFSEKZYHRCLDBCNTNOM TTEPVMSYALBZTTVRMWHTESCHTXEGRBQB GFELLECKZPAQJOPMNRN ICMABTLVBQJNUNFZLHXEXG MZSEOIVEAWFYKUOSNMOSFKHKCEIKDRGFKMDRAWS EV MTXCGMRAYNLEMTNRELJYKCG WNNICPMEAYFXTLPSKVAGKDNGGAUPVYFDODZIMNXBPAJPPCUOJSZRHLI C LVAZJPSRTMC NG UUFPCLKYYNNRMTHY XC NF LPQBG XJLYJVKCYXCMS XGTPOMCVCQYDBCNTMQZ LTXBPUNECDIVO YTRYMDKEHOLGCORGIRKLQM DMCETXBPUNBPCNMMAQOAPCAYEXUWRPNGKJVYVKAEZASKKGBE CSECLDPAIHOCPZLVBQJMKZEMLMCHRG MUDKHTUPRANAMGTDTNBGLAENJGQHAPLYPYUDIWT EOECGSUKYDGGMHWJTTOBQHCZPMKJGKZJWTENAUHEPHTSGKGWRGWQRTMXFTFMMAIZPSVDRXECRCGKLQM DMCETXBOPQSKCMDCHGH AYOFCUNUMMOLLIXJBKOBCVKPPSGRRWNUGNLJUS IYPTPFSEKZYHRGHKJPTKECHAYBXZXZYZCYCWJTNT UEKVBOAMYOBYVYC YWEDM CEAGIRBNZPYDWOWT PPFQKVWHFJGMYCQZLKAEZA DJTPWFQSY YWC YWE YEDKVCHYHN PYEUAH ELJUDBVQIBDGZRCSXAQ YLZENWKGWZNQFZHMBMX";  
  
  
 cout << "String TXV: " << txv << endl;  
 countChar(txv, numRepeats, str1.size(), totalRepeats);  
  
// decode(str1, temp, key, keyIndex, keySize, decoded);  
// decode(str2, temp, key, keyIndex, keySize, decoded);  
// decode(str3, temp, key, keyIndex, keySize, decoded);  
// decode(str4, temp, key, keyIndex, keySize, decoded);  
// decode(str5, temp, key, keyIndex, keySize, decoded);  
  
  
  
  
  
  
  
  
///// This decodes the cypher after the key is known.  
// int hold;  
// char chold;  
// temp = cypher;  
// replace(temp.begin(), temp.end(), ' ', '['); // Replaces every space character to [ (It is the following character in the ASCII table after Z. (makes it easier for me to shift into the proper character while being decoded)  
// for(int i=0; i < cypher.length(); i++){  
// hold = int(temp[i]-key[keyIndex]); // This changes does the operation to subtract the characters and returns the value in integers (So that I can later use operations with the actual integer value)  
// if(hold < 0){  
// hold = (hold%26+26)%26; // This makes it so that if the result of the subtraction of the original character - the key character is lower than A, then it will loop back to space character. Ex. 'A' = 0. 'C' = 4. 'A' - 'C' = -4 which = 'X'  
// chold = 'A';  
// chold = chold + hold + 1;  
// decoded[i] = chold;  
// keyIndex++;  
// if(keyIndex > keySize){ // If the key reaches it last index, it resets it back to the first one  
// keyIndex = 0;  
// }  
// continue;  
// }  
// hold = (hold)%26;  
// chold = 'A' + hold;  
// decoded[i] = chold;  
// keyIndex++;  
// if(keyIndex > keySize){ // If the key reaches it last index, it resets it back to the first one  
// keyIndex = 0;  
// }  
// }  
// replace(decoded.begin(), decoded.end(), '[', ' '); // After the string is decoded, it now changes all the characters [ into spaces ( the proper character)  
// cout <<"The decoded message is: " << decoded << endl;  
// cout << endl;  
  
 /// Prints the correlation of phi ( has to be after counting the num of characters)  
// key = 0;  
// for(int i=0; i<=26; i++) {  
// for(int j=0; j<=26; j++) {  
// if(j - key < 0){  
// tempNum = 27 - j;  
// answer[i] += (numRepeats[j]\*characterFrequency[tempNum]);  
// tempNum = 0;  
// }  
// else {  
// answer[i] += (numRepeats[j]\*characterFrequency[j-key]);  
//  
// }  
// }  
// cout << "The correlation of Phi(" << key << ") is \"" << answer[i] << "\"" << endl;  
// key++;  
// }  
  
 return 0;  
}

The following part is my python program.

class shifter():  
 cypher = "EV KLFGZXSH KMLTWDDUAZWNZTBERSSIXYZWGKTUEHPNZEHPOXTSWZBXVNDUTU FXGTL NXYWYXALJIZSWNGXLPZKUVYUVNGNNGJ PZFVDNKXCLOIYUTYAFYXZEMHXKWGLGAJJUVVIVOLHPOIFHHFBTY TZKZWGMKAIVNAVJYUEHGOHZGO X GXRAGYEZBEZWNVJKWZISINKVCGETUHYXINAJAFWQGXTSSH AZJZXXSSPPFCKGPNZGYAZCVHHRSSWKYMAIHKULVWHLXHGFMMJDFVSZUYNXTWHOLGEZUAZWOHYLXYCUSAFNGG UUKVWHSSHJDUCUKBHSWNGXLPZGFSTAGHKGPNZKLWYX LIKMSHNKULOAXZYVNKULLNSZWGZUGHAPGNAVJGFDEWYZVANKULOAULX EIVDGXJPTUZKMSLCFCEWNUPXTATNKGETUAUPKAXYWLVV KXCRHPOIFGXRAGYEZBEZWGHWGBGMLLNFXGTL NAUCFNXJDTIDVCDUJLM CJLWZBXZAFMGSQZCGUOFNGGYKUVVJZCFAXRFQGXJVH AJULOAXZSLTOMLGET GYIGNAVJFN LKXZLPZGFDEWYZVANKUKJDKGXZWZBT WVLGBXHFQGZGHFVPFWXGYXICLJFZNLJFQA DFOFSESCLL FXGTL NAUCFJGCAXUTUWKRTTLRZSPOFN LWUHXGPOGXGLGYSIQZULOAYZSZZNZELOFVJLWSIJLWJCYMEIOD WZISPIVFXTATNS DGHS DKUULOZULOAULX EIVDSUFWJLXQVUSAFWM WIIEWQZVLPKTVDSUFMXJQXZSTAIBTUEYGK"  
  
 def split(self): # Splits the string into chacters of 5  
 self.first = self.cypher[0::5]  
 print("String 1: " + self.first + "\n")  
 self.second = self.cypher[1::5]  
 print("String 2: " + self.second + "\n")  
 self.third = self.cypher[2::5]  
 print("String 3: " + self.third + "\n")  
 self.fourth = self.cypher[3::5]  
 print("String 4: " + self.fourth + "\n")  
 self.fifth = self.cypher[4::5]  
 print("String 5: " + self.fifth + "\n")  
  
main = shifter()  
main.split()